Jon Postel

Network Working Group
Request for Comments: 1122

Internet Engineering Task Force
R. Braden, Editor
October 1989

Requirements for Internet Hosts -- Communication Layers

Status of This Memo

This RFC is an official specification for the Internet community. It incorporates by reference, amends, corrects, and supplements the primary protocol standards documents relating to hosts. Distribution of this document is unlimited.

Summary

This is one RFC of a pair that defines and discusses the requirements for Internet host software. This RFC covers the communications protocol layers: link layer, IP layer, and transport layer; its companion RFC-1123 covers the application and support protocols.

Table of Contents

1. INTRODUCTION 5
  1.1 The Internet Architecture 6
    1.1.1 Internet Hosts 6
    1.1.2 Architectural Assumptions 7
    1.1.3 Internet Protocol Suite 8
    1.1.4 Embedded Gateway Code 10
  1.2 General Considerations 12
    1.2.1 Continuing Internet Evolution 12
    1.2.2 Robustness Principle 12
    1.2.3 Error Logging 13
    1.2.4 Configuration 14
  1.3 Reading this Document 15
    1.3.1 Organization 15
    1.3.2 Requirements 16
    1.3.3 Terminology 17
  1.4 Acknowledgments 20

2. LINK LAYER 21
  2.1 INTRODUCTION 21

2.2 PROTOCOL WALK-THROUGH 21
2.3 SPECIFIC ISSUES 21
  2.3.1 Trailer Protocol Negotiation 21
  2.3.2 Address Resolution Protocol -- ARP 22
    2.3.2.1 ARP Cache Validation 22
    2.3.2.2 ARP Packet Queue 24
  2.3.3 Ethernet and IEEE 802 Encapsulation 24
2.4 LINK/INTERNET LAYER INTERFACE 25
2.5 LINK LAYER REQUIREMENTS SUMMARY 26

3. INTERNET LAYER PROTOCOLS 27
3.1 INTRODUCTION 27
3.2 PROTOCOL WALK-THROUGH 29
  3.2.1 Internet Protocol -- IP 29
    3.2.1.1 Version Number 29
    3.2.1.2 Checksum 29
    3.2.1.3 Addressing 29
    3.2.1.4 Fragmentation and Reassembly 32
    3.2.1.5 Identification 32
    3.2.1.6 Type-of-Service 33
    3.2.1.7 Time-to-Live 34
    3.2.1.8 Options 35
  3.2.2 Internet Control Message Protocol -- ICMP 38
    3.2.2.1 Destination Unreachable 39
    3.2.2.2 Redirect 40
    3.2.2.3 Source Quench 41
    3.2.2.4 Time Exceeded 41
    3.2.2.5 Parameter Problem 42
    3.2.2.6 Echo Request/Reply 42
    3.2.2.7 Information Request/Reply 43
    3.2.2.8 Timestamp and Timestamp Reply 43
    3.2.2.9 Address Mask Request/Reply 45
  3.2.3 Internet Group Management Protocol IGMP 47
3.3 SPECIFIC ISSUES 47
  3.3.1 Routing Outbound Datagrams 47
    3.3.1.1 Local/Remote Decision 47
    3.3.1.2 Gateway Selection 48
    3.3.1.3 Route Cache 49
    3.3.1.4 Dead Gateway Detection 51
    3.3.1.5 New Gateway Selection 55
    3.3.1.6 Initialization 56
  3.3.2 Reassembly 56
  3.3.3 Fragmentation 58
  3.3.4 Local Multihoming 60
    3.3.4.1 Introduction 60
    3.3.4.2 Multihoming Requirements 61
    3.3.4.3 Choosing a Source Address 64
  3.3.5 Source Route Forwarding 65

3.3.6 Broadcasts 66
3.3.7 IP Multicasting 67
3.3.8 Error Reporting 69
3.4 INTERNET/TRANSPORT LAYER INTERFACE 69
3.5 INTERNET LAYER REQUIREMENTS SUMMARY 72

4. TRANSPORT PROTOCOLS 77
4.1 USER DATAGRAM PROTOCOL -- UDP 77
4.1.1 INTRODUCTION 77
4.1.2 PROTOCOL WALK-THROUGH 77
4.1.3 SPECIFIC ISSUES 77
4.1.3.1 Ports 77
4.1.3.2 IP Options 77
4.1.3.3 ICMP Messages 78
4.1.3.4 UDP Checksums 78
4.1.3.5 UDP Multihoming 79
4.1.3.6 Invalid Addresses 79
4.1.4 UDP/APPLICATION LAYER INTERFACE 79
4.1.5 UDP REQUIREMENTS SUMMARY 80
4.2 TRANSMISSION CONTROL PROTOCOL -- TCP 82
4.2.1 INTRODUCTION 82
4.2.2 PROTOCOL WALK-THROUGH 82
4.2.2.1 Well-Known Ports 82
4.2.2.2 Use of Push 82
4.2.2.3 Window Size 83
4.2.2.4 Urgent Pointer 84
4.2.2.5 TCP Options 85
4.2.2.6 Maximum Segment Size Option 85
4.2.2.7 TCP Checksum 86
4.2.2.8 TCP Connection State Diagram 86
4.2.2.9 Initial Sequence Number Selection 87
4.2.2.10 Simultaneous Open Attempts 87
4.2.2.11 Recovery from Old Duplicate SYN 87
4.2.2.12 RST Segment 87
4.2.2.13 Closing a Connection 87
4.2.2.14 Data Communication 89
4.2.2.15 Retransmission Timeout 90
4.2.2.16 Managing the Window 91
4.2.2.17 Probing Zero Windows 92
4.2.2.18 Passive OPEN Calls 92
4.2.2.19 Time to Live 93
4.2.2.20 Event Processing 93
4.2.2.21 Acknowledging Queued Segments 94
4.2.3 SPECIFIC ISSUES 95
4.2.3.1 Retransmission Timeout Calculation 95
4.2.3.2 When to Send an ACK Segment 96
4.2.3.3 When to Send a Window Update 97
4.2.3.4 When to Send Data 98

    4.2.3.5 TCP Connection Failures 100
    4.2.3.6 TCP Keep-Alives 101
    4.2.3.7 TCP Multihoming 103
    4.2.3.8 IP Options 103
    4.2.3.9 ICMP Messages 103
    4.2.3.10 Remote Address Validation 104
    4.2.3.11 TCP Traffic Patterns 104
    4.2.3.12 Efficiency 105
  4.2.4 TCP/APPLICATION LAYER INTERFACE 106
    4.2.4.1 Asynchronous Reports 106
    4.2.4.2 Type-of-Service 107
    4.2.4.3 Flush Call 107
    4.2.4.4 Multihoming 108
  4.2.5 TCP REQUIREMENT SUMMARY 108

5. REFERENCES 112

1. INTRODUCTION

This document is one of a pair that defines and discusses the requirements for host system implementations of the Internet protocol suite. This RFC covers the communication protocol layers: link layer, IP layer, and transport layer. Its companion RFC, "Requirements for Internet Hosts -- Application and Support" [INTRO:1], covers the application layer protocols. This document should also be read in conjunction with "Requirements for Internet Gateways" [INTRO:2].

These documents are intended to provide guidance for vendors, implementors, and users of Internet communication software. They represent the consensus of a large body of technical experience and wisdom, contributed by the members of the Internet research and vendor communities.

This RFC enumerates standard protocols that a host connected to the Internet must use, and it incorporates by reference the RFCs and other documents describing the current specifications for these protocols. It corrects errors in the referenced documents and adds additional discussion and guidance for an implementor.

For each protocol, this document also contains an explicit set of requirements, recommendations, and options. The reader must understand that the list of requirements in this document is incomplete by itself; the complete set of requirements for an Internet host is primarily defined in the standard protocol specification documents, with the corrections, amendments, and supplements contained in this RFC.

A good-faith implementation of the protocols that was produced after careful reading of the RFC's and with some interaction with the Internet technical community, and that followed good communications software engineering practices, should differ from the requirements of this document in only minor ways. Thus, in many cases, the "requirements" in this RFC are already stated or implied in the standard protocol documents, so that their inclusion here is, in a sense, redundant. However, they were included because some past implementation has made the wrong choice, causing problems of interoperability, performance, and/or robustness.

This document includes discussion and explanation of many of the requirements and recommendations. A simple list of requirements would be dangerous, because:

- o Some required features are more important than others, and some features are optional.

- o There may be valid reasons why particular vendor products that are designed for restricted contexts might choose to use different specifications.

However, the specifications of this document must be followed to meet the general goal of arbitrary host interoperation across the diversity and complexity of the Internet system. Although most current implementations fail to meet these requirements in various ways, some minor and some major, this specification is the ideal towards which we need to move.

These requirements are based on the current level of Internet architecture. This document will be updated as required to provide additional clarifications or to include additional information in those areas in which specifications are still evolving.

This introductory section begins with a brief overview of the Internet architecture as it relates to hosts, and then gives some general advice to host software vendors. Finally, there is some guidance on reading the rest of the document and some terminology.

1.1 The Internet Architecture

General background and discussion on the Internet architecture and supporting protocol suite can be found in the DDN Protocol Handbook [INTRO:3]; for background see for example [INTRO:9], [INTRO:10], and [INTRO:11]. Reference [INTRO:5] describes the procedure for obtaining Internet protocol documents, while [INTRO:6] contains a list of the numbers assigned within Internet protocols.

1.1.1 Internet Hosts

A host computer, or simply "host," is the ultimate consumer of communication services. A host generally executes application programs on behalf of user(s), employing network and/or Internet communication services in support of this function. An Internet host corresponds to the concept of an "End-System" used in the OSI protocol suite [INTRO:13].

An Internet communication system consists of interconnected packet networks supporting communication among host computers using the Internet protocols. The networks are interconnected using packet-switching computers called "gateways" or "IP routers" by the Internet community, and "Intermediate Systems" by the OSI world [INTRO:13]. The RFC "Requirements for Internet Gateways" [INTRO:2] contains the official specifications for Internet gateways. That RFC together with the present document and its companion [INTRO:1] define the rules for the current realization of the Internet architecture.

Internet hosts span a wide range of size, speed, and function. They range in size from small microprocessors through workstations to mainframes and supercomputers. In function, they range from single-purpose hosts (such as terminal servers) to full-service hosts that support a variety of online network services, typically including remote login, file transfer, and electronic mail.

A host is generally said to be multihomed if it has more than one interface to the same or to different networks. See Section 1.1.3 on "Terminology".

1.1.2 Architectural Assumptions

The current Internet architecture is based on a set of assumptions about the communication system. The assumptions most relevant to hosts are as follows:

(a) The Internet is a network of networks.

Each host is directly connected to some particular network(s); its connection to the Internet is only conceptual. Two hosts on the same network communicate with each other using the same set of protocols that they would use to communicate with hosts on distant networks.

(b) Gateways don't keep connection state information.

To improve robustness of the communication system, gateways are designed to be stateless, forwarding each IP datagram independently of other datagrams. As a result, redundant paths can be exploited to provide robust service in spite of failures of intervening gateways and networks.

All state information required for end-to-end flow control and reliability is implemented in the hosts, in the transport layer or in application programs. All connection control information is thus co-located with the end points of the communication, so it will be lost only if an end point fails.

(c) Routing complexity should be in the gateways.

Routing is a complex and difficult problem, and ought to be performed by the gateways, not the hosts. An important objective is to insulate host software from changes caused by the inevitable evolution of the Internet routing architecture.

(d) The System must tolerate wide network variation.

A basic objective of the Internet design is to tolerate a wide range of network characteristics -- e.g., bandwidth, delay, packet loss, packet reordering, and maximum packet size. Another objective is robustness against failure of individual networks, gateways, and hosts, using whatever bandwidth is still available. Finally, the goal is full "open system interconnection": an Internet host must be able to interoperate robustly and effectively with any other Internet host, across diverse Internet paths.

Sometimes host implementors have designed for less ambitious goals. For example, the LAN environment is typically much more benign than the Internet as a whole; LANs have low packet loss and delay and do not reorder packets. Some vendors have fielded host implementations that are adequate for a simple LAN environment, but work badly for general interoperation. The vendor justifies such a product as being economical within the restricted LAN market. However, isolated LANs seldom stay isolated for long; they are soon gatewayed to each other, to organization-wide internets, and eventually to the global Internet system. In the end, neither the customer nor the vendor is served by incomplete or substandard Internet host software.

The requirements spelled out in this document are designed for a full-function Internet host, capable of full interoperation over an arbitrary Internet path.

1.1.3 Internet Protocol Suite

To communicate using the Internet system, a host must implement the layered set of protocols comprising the Internet protocol suite. A host typically must implement at least one protocol from each layer.

The protocol layers used in the Internet architecture are as follows [INTRO:4]:

o Application Layer

The application layer is the top layer of the Internet protocol suite. The Internet suite does not further subdivide the application layer, although some of the Internet application layer protocols do contain some internal sub-layering. The application layer of the Internet suite essentially combines the functions of the top two layers -- Presentation and Application -- of the OSI reference model.

We distinguish two categories of application layer protocols: user protocols that provide service directly to users, and support protocols that provide common system functions. Requirements for user and support protocols will be found in the companion RFC [INTRO:1].

The most common Internet user protocols are:

o Telnet	(remote login)
o FTP	(file transfer)
o SMTP	(electronic mail delivery)

There are a number of other standardized user protocols [INTRO:4] and many private user protocols.

Support protocols, used for host name mapping, booting, and management, include SNMP, BOOTP, RARP, and the Domain Name System (DNS) protocols.

o Transport Layer

The transport layer provides end-to-end communication services for applications. There are two primary transport layer protocols at present:

- o Transmission Control Protocol (TCP)
- o User Datagram Protocol (UDP)

TCP is a reliable connection-oriented transport service that provides end-to-end reliability, resequencing, and flow control. UDP is a connectionless ("datagram") transport service.

Other transport protocols have been developed by the research community, and the set of official Internet transport protocols may be expanded in the future.

Transport layer protocols are discussed in Chapter 4.

- Internet Layer

  All Internet transport protocols use the Internet Protocol (IP) to carry data from source host to destination host. IP is a connectionless or datagram internetwork service, providing no end-to-end delivery guarantees. Thus, IP datagrams may arrive at the destination host damaged, duplicated, out of order, or not at all. The layers above IP are responsible for reliable delivery service when it is required. The IP protocol includes provision for addressing, type-of-service specification, fragmentation and reassembly, and security information.

  The datagram or connectionless nature of the IP protocol is a fundamental and characteristic feature of the Internet architecture. Internet IP was the model for the OSI Connectionless Network Protocol [INTRO:12].

  ICMP is a control protocol that is considered to be an integral part of IP, although it is architecturally layered upon IP, i.e., it uses IP to carry its data end-to-end just as a transport protocol like TCP or UDP does. ICMP provides error reporting, congestion reporting, and first-hop gateway redirection.

  IGMP is an Internet layer protocol used for establishing dynamic host groups for IP multicasting.

  The Internet layer protocols IP, ICMP, and IGMP are discussed in Chapter 3.

- Link Layer

  To communicate on its directly-connected network, a host must implement the communication protocol used to interface to that network. We call this a link layer or media-access layer protocol.

  There is a wide variety of link layer protocols, corresponding to the many different types of networks. See Chapter 2.

1.1.4 Embedded Gateway Code

Some Internet host software includes embedded gateway functionality, so that these hosts can forward packets as a gateway would, while still performing the application layer functions of a host.

Such dual-purpose systems must follow the Gateway Requirements RFC [INTRO:2] with respect to their gateway functions, and must follow the present document with respect to their host functions. In all overlapping cases, the two specifications should be in agreement.

There are varying opinions in the Internet community about embedded gateway functionality. The main arguments are as follows:

o Pro: in a local network environment where networking is informal, or in isolated internets, it may be convenient and economical to use existing host systems as gateways.

  There is also an architectural argument for embedded gateway functionality: multihoming is much more common than originally foreseen, and multihoming forces a host to make routing decisions as if it were a gateway. If the multihomed host contains an embedded gateway, it will have full routing knowledge and as a result will be able to make more optimal routing decisions.

o Con: Gateway algorithms and protocols are still changing, and they will continue to change as the Internet system grows larger. Attempting to include a general gateway function within the host IP layer will force host system maintainers to track these (more frequent) changes. Also, a larger pool of gateway implementations will make coordinating the changes more difficult. Finally, the complexity of a gateway IP layer is somewhat greater than that of a host, making the implementation and operation tasks more complex.

  In addition, the style of operation of some hosts is not appropriate for providing stable and robust gateway service.

There is considerable merit in both of these viewpoints. One conclusion can be drawn: an host administrator must have conscious control over whether or not a given host acts as a gateway. See Section 3.1 for the detailed requirements.

1.2 General Considerations

There are two important lessons that vendors of Internet host software have learned and which a new vendor should consider seriously.

1.2.1 Continuing Internet Evolution

The enormous growth of the Internet has revealed problems of management and scaling in a large datagram-based packet communication system. These problems are being addressed, and as a result there will be continuing evolution of the specifications described in this document. These changes will be carefully planned and controlled, since there is extensive participation in this planning by the vendors and by the organizations responsible for operations of the networks.

Development, evolution, and revision are characteristic of computer network protocols today, and this situation will persist for some years. A vendor who develops computer communication software for the Internet protocol suite (or any other protocol suite!) and then fails to maintain and update that software for changing specifications is going to leave a trail of unhappy customers. The Internet is a large communication network, and the users are in constant contact through it. Experience has shown that knowledge of deficiencies in vendor software propagates quickly through the Internet technical community.

1.2.2 Robustness Principle

At every layer of the protocols, there is a general rule whose application can lead to enormous benefits in robustness and interoperability [IP:1]:

> "Be liberal in what you accept, and
> conservative in what you send"

Software should be written to deal with every conceivable error, no matter how unlikely; sooner or later a packet will come in with that particular combination of errors and attributes, and unless the software is prepared, chaos can ensue. In general, it is best to assume that the network is filled with malevolent entities that will send in packets designed to have the worst possible effect. This assumption will lead to suitable protective design, although the most serious problems in the Internet have been caused by unenvisaged mechanisms triggered by low-probability events;

mere human malice would never have taken so devious a course!

Adaptability to change must be designed into all levels of Internet host software. As a simple example, consider a protocol specification that contains an enumeration of values for a particular header field -- e.g., a type field, a port number, or an error code; this enumeration must be assumed to be incomplete. Thus, if a protocol specification defines four possible error codes, the software must not break when a fifth code shows up. An undefined code might be logged (see below), but it must not cause a failure.

The second part of the principle is almost as important: software on other hosts may contain deficiencies that make it unwise to exploit legal but obscure protocol features. It is unwise to stray far from the obvious and simple, lest untoward effects result elsewhere. A corollary of this is "watch out for misbehaving hosts"; host software should be prepared, not just to survive other misbehaving hosts, but also to cooperate to limit the amount of disruption such hosts can cause to the shared communication facility.

1.2.3 Error Logging

The Internet includes a great variety of host and gateway systems, each implementing many protocols and protocol layers, and some of these contain bugs and mis-features in their Internet protocol software. As a result of complexity, diversity, and distribution of function, the diagnosis of Internet problems is often very difficult.

Problem diagnosis will be aided if host implementations include a carefully designed facility for logging erroneous or "strange" protocol events. It is important to include as much diagnostic information as possible when an error is logged. In particular, it is often useful to record the header(s) of a packet that caused an error. However, care must be taken to ensure that error logging does not consume prohibitive amounts of resources or otherwise interfere with the operation of the host.

There is a tendency for abnormal but harmless protocol events to overflow error logging files; this can be avoided by using a "circular" log, or by enabling logging only while diagnosing a known failure. It may be useful to filter and count duplicate successive messages. One strategy that seems to work well is: (1) always count abnormalities and make such counts accessible through the management protocol (see [INTRO:1]); and (2) allow the logging of a great variety of events to be selectively enabled. For example, it might useful to be able to "log everything" or to "log everything for host X".

Note that different managements may have differing policies about the amount of error logging that they want normally enabled in a host. Some will say, "if it doesn't hurt me, I don't want to know about it", while others will want to take a more watchful and aggressive attitude about detecting and removing protocol abnormalities.

1.2.4 Configuration

It would be ideal if a host implementation of the Internet protocol suite could be entirely self-configuring. This would allow the whole suite to be implemented in ROM or cast into silicon, it would simplify diskless workstations, and it would be an immense boon to harried LAN administrators as well as system vendors. We have not reached this ideal; in fact, we are not even close.

At many points in this document, you will find a requirement that a parameter be a configurable option. There are several different reasons behind such requirements. In a few cases, there is current uncertainty or disagreement about the best value, and it may be necessary to update the recommended value in the future. In other cases, the value really depends on external factors -- e.g., the size of the host and the distribution of its communication load, or the speeds and topology of nearby networks -- and self-tuning algorithms are unavailable and may be insufficient. In some cases, configurability is needed because of administrative requirements.

Finally, some configuration options are required to communicate with obsolete or incorrect implementations of the protocols, distributed without sources, that unfortunately persist in many parts of the Internet. To make correct systems coexist with these faulty systems, administrators often have to "mis-configure" the correct systems. This problem will correct itself gradually as the faulty systems are retired, but it cannot be ignored by vendors.

When we say that a parameter must be configurable, we do not intend to require that its value be explicitly read from a configuration file at every boot time. We recommend that implementors set up a default for each parameter, so a configuration file is only necessary to override those defaults that are inappropriate in a particular installation. Thus, the configurability requirement is an assurance that it will be POSSIBLE to override the default when necessary, even in a binary-only or ROM-based product.

This document requires a particular value for such defaults in some cases. The choice of default is a sensitive issue when the configuration item controls the accommodation to existing faulty systems. If the Internet is to converge successfully to complete interoperability, the default values built into implementations must implement the official protocol, not "mis-configurations" to accommodate faulty implementations. Although marketing considerations have led some vendors to choose mis-configuration defaults, we urge vendors to choose defaults that will conform to the standard.

Finally, we note that a vendor needs to provide adequate documentation on all configuration parameters, their limits and effects.

1.3 Reading this Document

1.3.1 Organization

Protocol layering, which is generally used as an organizing principle in implementing network software, has also been used to organize this document. In describing the rules, we assume that an implementation does strictly mirror the layering of the protocols. Thus, the following three major sections specify the requirements for the link layer, the internet layer, and the transport layer, respectively. A companion RFC [INTRO:1] covers application level software. This layerist organization was chosen for simplicity and clarity.

However, strict layering is an imperfect model, both for the protocol suite and for recommended implementation approaches. Protocols in different layers interact in complex and sometimes subtle ways, and particular functions often involve multiple layers. There are many design choices in an implementation, many of which involve creative "breaking" of strict layering. Every implementor is urged to read references [INTRO:7] and [INTRO:8].

This document describes the conceptual service interface between layers using a functional ("procedure call") notation, like that used in the TCP specification [TCP:1]. A host implementation must support the logical information flow implied by these calls, but need not literally implement the calls themselves. For example, many implementations reflect the coupling between the transport layer and the IP layer by giving them shared access to common data structures. These data structures, rather than explicit procedure calls, are then the agency for passing much of the information that is required.

In general, each major section of this document is organized into the following subsections:

(1) Introduction

(2) Protocol Walk-Through -- considers the protocol specification documents section-by-section, correcting errors, stating requirements that may be ambiguous or ill-defined, and providing further clarification or explanation.

(3) Specific Issues -- discusses protocol design and implementation issues that were not included in the walk-through.

(4) Interfaces -- discusses the service interface to the next higher layer.

(5) Summary -- contains a summary of the requirements of the section.

Under many of the individual topics in this document, there is parenthetical material labeled "DISCUSSION" or "IMPLEMENTATION". This material is intended to give clarification and explanation of the preceding requirements text. It also includes some suggestions on possible future directions or developments. The implementation material contains suggested approaches that an implementor may want to consider.

The summary sections are intended to be guides and indexes to the text, but are necessarily cryptic and incomplete. The summaries should never be used or referenced separately from the complete RFC.

1.3.2 Requirements

In this document, the words that are used to define the significance of each particular requirement are capitalized.

These words are:

- "MUST"

  This word or the adjective "REQUIRED" means that the item is an absolute requirement of the specification.

- "SHOULD"

  This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

- "MAY"

  This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for the protocols it implements. An implementation that satisfies all the MUST and all the SHOULD requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST requirements but not all the SHOULD requirements for its protocols is said to be "conditionally compliant".

1.3.3 Terminology

This document uses the following technical terms:

Segment
A segment is the unit of end-to-end transmission in the TCP protocol. A segment consists of a TCP header followed by application data. A segment is transmitted by encapsulation inside an IP datagram.

Message
In this description of the lower-layer protocols, a message is the unit of transmission in a transport layer protocol. In particular, a TCP segment is a message. A message consists of a transport protocol header followed by application protocol data. To be transmitted end-to-

end through the Internet, a message must be encapsulated inside a datagram.

IP Datagram

An IP datagram is the unit of end-to-end transmission in the IP protocol. An IP datagram consists of an IP header followed by transport layer data, i.e., of an IP header followed by a message.

In the description of the internet layer (Section 3), the unqualified term "datagram" should be understood to refer to an IP datagram.

Packet

A packet is the unit of data passed across the interface between the internet layer and the link layer. It includes an IP header and data. A packet may be a complete IP datagram or a fragment of an IP datagram.

Frame

A frame is the unit of transmission in a link layer protocol, and consists of a link-layer header followed by a packet.

Connected Network

A network to which a host is interfaced is often known as the "local network" or the "subnetwork" relative to that host. However, these terms can cause confusion, and therefore we use the term "connected network" in this document.

Multihomed

A host is said to be multihomed if it has multiple IP addresses. For a discussion of multihoming, see Section 3.3.4 below.

Physical network interface

This is a physical interface to a connected network and has a (possibly unique) link-layer address. Multiple physical network interfaces on a single host may share the same link-layer address, but the address must be unique for different hosts on the same physical network.

Logical [network] interface

We define a logical [network] interface to be a logical path, distinguished by a unique IP address, to a connected network. See Section 3.3.4.

Specific-destination address
This is the effective destination address of a datagram, even if it is broadcast or multicast; see Section 3.2.1.3.

Path
At a given moment, all the IP datagrams from a particular source host to a particular destination host will typically traverse the same sequence of gateways. We use the term "path" for this sequence. Note that a path is uni-directional; it is not unusual to have different paths in the two directions between a given host pair.

MTU
The maximum transmission unit, i.e., the size of the largest packet that can be transmitted.

The terms frame, packet, datagram, message, and segment are illustrated by the following schematic diagrams:

A. Transmission on connected network:

```
 _______________________________________________
| LL hdr | IP hdr |         (data)              |
|________|________|_____________________________|

 <---------- Frame ----------------------------->
          <----------Packet -------------------->
```

B. Before IP fragmentation or after IP reassembly:

```
          ______________________________________
         | IP hdr | transport| Application Data |
         |________|____hdr___|__________________|

          <--------  Datagram ------------------>
                   <-------- Message ----------->
```

or, for TCP:

```
          ______________________________________
         | IP hdr |  TCP hdr | Application Data |
         |________|__________|__________________|

          <--------  Datagram ------------------>
                   <-------- Segment ----------->
```

1.4 Acknowledgments

This document incorporates contributions and comments from a large group of Internet protocol experts, including representatives of university and research labs, vendors, and government agencies. It was assembled primarily by the Host Requirements Working Group of the Internet Engineering Task Force (IETF).

The Editor would especially like to acknowledge the tireless dedication of the following people, who attended many long meetings and generated 3 million bytes of electronic mail over the past 18 months in pursuit of this document: Philip Almquist, Dave Borman (Cray Research), Noel Chiappa, Dave Crocker (DEC), Steve Deering (Stanford), Mike Karels (Berkeley), Phil Karn (Bellcore), John Lekashman (NASA), Charles Lynn (BBN), Keith McCloghrie (TWG), Paul Mockapetris (ISI), Thomas Narten (Purdue), Craig Partridge (BBN), Drew Perkins (CMU), and James Van Bokkelen (FTP Software).

In addition, the following people made major contributions to the effort: Bill Barns (Mitre), Steve Bellovin (AT&T), Mike Brescia (BBN), Ed Cain (DCA), Annette DeSchon (ISI), Martin Gross (DCA), Phill Gross (NRI), Charles Hedrick (Rutgers), Van Jacobson (LBL), John Klensin (MIT), Mark Lottor (SRI), Milo Medin (NASA), Bill Melohn (Sun Microsystems), Greg Minshall (Kinetics), Jeff Mogul (DEC), John Mullen (CMC), Jon Postel (ISI), John Romkey (Epilogue Technology), and Mike StJohns (DCA). The following also made significant contributions to particular areas: Eric Allman (Berkeley), Rob Austein (MIT), Art Berggreen (ACC), Keith Bostic (Berkeley), Vint Cerf (NRI), Wayne Hathaway (NASA), Matt Korn (IBM), Erik Naggum (Naggum Software, Norway), Robert Ullmann (Prime Computer), David Waitzman (BBN), Frank Wancho (USA), Arun Welch (Ohio State), Bill Westfield (Cisco), and Rayan Zachariassen (Toronto).

We are grateful to all, including any contributors who may have been inadvertently omitted from this list.

2. LINK LAYER

2.1 INTRODUCTION

All Internet systems, both hosts and gateways, have the same requirements for link layer protocols. These requirements are given in Chapter 3 of "Requirements for Internet Gateways" [INTRO:2], augmented with the material in this section.

2.2 PROTOCOL WALK-THROUGH

None.

2.3 SPECIFIC ISSUES

2.3.1 Trailer Protocol Negotiation

The trailer protocol [LINK:1] for link-layer encapsulation MAY be used, but only when it has been verified that both systems (host or gateway) involved in the link-layer communication implement trailers. If the system does not dynamically negotiate use of the trailer protocol on a per-destination basis, the default configuration MUST disable the protocol.

DISCUSSION:

The trailer protocol is a link-layer encapsulation technique that rearranges the data contents of packets sent on the physical network. In some cases, trailers improve the throughput of higher layer protocols by reducing the amount of data copying within the operating system. Higher layer protocols are unaware of trailer use, but both the sending and receiving host MUST understand the protocol if it is used.

Improper use of trailers can result in very confusing symptoms. Only packets with specific size attributes are encapsulated using trailers, and typically only a small fraction of the packets being exchanged have these attributes. Thus, if a system using trailers exchanges packets with a system that does not, some packets disappear into a black hole while others are delivered successfully.

IMPLEMENTATION:

On an Ethernet, packets encapsulated with trailers use a distinct Ethernet type [LINK:1], and trailer negotiation is performed at the time that ARP is used to discover the link-layer address of a destination system.

Specifically, the ARP exchange is completed in the usual manner using the normal IP protocol type, but a host that wants to speak trailers will send an additional "trailer ARP reply" packet, i.e., an ARP reply that specifies the trailer encapsulation protocol type but otherwise has the format of a normal ARP reply. If a host configured to use trailers receives a trailer ARP reply message from a remote machine, it can add that machine to the list of machines that understand trailers, e.g., by marking the corresponding entry in the ARP cache.

Hosts wishing to receive trailer encapsulations send trailer ARP replies whenever they complete exchanges of normal ARP messages for IP. Thus, a host that received an ARP request for its IP protocol address would send a trailer ARP reply in addition to the normal IP ARP reply; a host that sent the IP ARP request would send a trailer ARP reply when it received the corresponding IP ARP reply. In this way, either the requesting or responding host in an IP ARP exchange may request that it receive trailer encapsulations.

This scheme, using extra trailer ARP reply packets rather than sending an ARP request for the trailer protocol type, was designed to avoid a continuous exchange of ARP packets with a misbehaving host that, contrary to any specification or common sense, responded to an ARP reply for trailers with another ARP reply for IP. This problem is avoided by sending a trailer ARP reply in response to an IP ARP reply only when the IP ARP reply answers an outstanding request; this is true when the hardware address for the host is still unknown when the IP ARP reply is received. A trailer ARP reply may always be sent along with an IP ARP reply responding to an IP ARP request.

2.3.2 Address Resolution Protocol -- ARP

2.3.2.1 ARP Cache Validation

An implementation of the Address Resolution Protocol (ARP) [LINK:2] MUST provide a mechanism to flush out-of-date cache entries. If this mechanism involves a timeout, it SHOULD be possible to configure the timeout value.

A mechanism to prevent ARP flooding (repeatedly sending an ARP Request for the same IP address, at a high rate) MUST be included. The recommended maximum rate is 1 per second per destination.

DISCUSSION:

The ARP specification [LINK:2] suggests but does not require a timeout mechanism to invalidate cache entries when hosts change their Ethernet addresses. The prevalence of proxy ARP (see Section 2.4 of [INTRO:2]) has significantly increased the likelihood that cache entries in hosts will become invalid, and therefore some ARP-cache invalidation mechanism is now required for hosts. Even in the absence of proxy ARP, a long-period cache timeout is useful in order to automatically correct any bad ARP data that might have been cached.

IMPLEMENTATION:

Four mechanisms have been used, sometimes in combination, to flush out-of-date cache entries.

(1) Timeout -- Periodically time out cache entries, even if they are in use. Note that this timeout should be restarted when the cache entry is "refreshed" (by observing the source fields, regardless of target address, of an ARP broadcast from the system in question). For proxy ARP situations, the timeout needs to be on the order of a minute.

(2) Unicast Poll -- Actively poll the remote host by periodically sending a point-to-point ARP Request to it, and delete the entry if no ARP Reply is received from N successive polls. Again, the timeout should be on the order of a minute, and typically N is 2.

(3) Link-Layer Advice -- If the link-layer driver detects a delivery problem, flush the corresponding ARP cache entry.

(4) Higher-layer Advice -- Provide a call from the Internet layer to the link layer to indicate a delivery problem. The effect of this call would be to invalidate the corresponding cache entry. This call would be analogous to the "ADVISE_DELIVPROB()" call from the transport layer to the Internet layer (see Section 3.4), and in fact the ADVISE_DELIVPROB routine might in turn call the link-layer advice routine to invalidate

the ARP cache entry.

Approaches (1) and (2) involve ARP cache timeouts on the order of a minute or less. In the absence of proxy ARP, a timeout this short could create noticeable overhead traffic on a very large Ethernet. Therefore, it may be necessary to configure a host to lengthen the ARP cache timeout.

2.3.2.2 ARP Packet Queue

The link layer SHOULD save (rather than discard) at least one (the latest) packet of each set of packets destined to the same unresolved IP address, and transmit the saved packet when the address has been resolved.

DISCUSSION:

Failure to follow this recommendation causes the first packet of every exchange to be lost. Although higher-layer protocols can generally cope with packet loss by retransmission, packet loss does impact performance. For example, loss of a TCP open request causes the initial round-trip time estimate to be inflated. UDP-based applications such as the Domain Name System are more seriously affected.

2.3.3 Ethernet and IEEE 802 Encapsulation

The IP encapsulation for Ethernets is described in RFC-894 [LINK:3], while RFC-1042 [LINK:4] describes the IP encapsulation for IEEE 802 networks. RFC-1042 elaborates and replaces the discussion in Section 3.4 of [INTRO:2].

Every Internet host connected to a 10Mbps Ethernet cable:

- o MUST be able to send and receive packets using RFC-894 encapsulation;
- o SHOULD be able to receive RFC-1042 packets, intermixed with RFC-894 packets; and
- o MAY be able to send packets using RFC-1042 encapsulation.

An Internet host that implements sending both the RFC-894 and the RFC-1042 encapsulations MUST provide a configuration switch to select which is sent, and this switch MUST default to RFC-894.

Note that the standard IP encapsulation in RFC-1042 does not use the protocol id value (K1=6) that IEEE reserved for IP; instead, it uses a value (K1=170) that implies an extension (the "SNAP") which can be used to hold the Ether-Type field. An Internet system MUST NOT send 802 packets using K1=6.

Address translation from Internet addresses to link-layer addresses on Ethernet and IEEE 802 networks MUST be managed by the Address Resolution Protocol (ARP).

The MTU for an Ethernet is 1500 and for 802.3 is 1492.

DISCUSSION:

The IEEE 802.3 specification provides for operation over a 10Mbps Ethernet cable, in which case Ethernet and IEEE 802.3 frames can be physically intermixed. A receiver can distinguish Ethernet and 802.3 frames by the value of the 802.3 Length field; this two-octet field coincides in the header with the Ether-Type field of an Ethernet frame. In particular, the 802.3 Length field must be less than or equal to 1500, while all valid Ether-Type values are greater than 1500.

Another compatibility problem arises with link-layer broadcasts. A broadcast sent with one framing will not be seen by hosts that can receive only the other framing.

The provisions of this section were designed to provide direct interoperation between 894-capable and 1042-capable systems on the same cable, to the maximum extent possible. It is intended to support the present situation where 894-only systems predominate, while providing an easy transition to a possible future in which 1042-capable systems become common.

Note that 894-only systems cannot interoperate directly with 1042-only systems. If the two system types are set up as two different logical networks on the same cable, they can communicate only through an IP gateway. Furthermore, it is not useful or even possible for a dual-format host to discover automatically which format to send, because of the problem of link-layer broadcasts.

2.4 LINK/INTERNET LAYER INTERFACE

The packet receive interface between the IP layer and the link layer MUST include a flag to indicate whether the incoming packet was addressed to a link-layer broadcast address.

DISCUSSION

Although the IP layer does not generally know link layer addresses (since every different network medium typically has a different address format), the broadcast address on a broadcast-capable medium is an important special case. See Section 3.2.2, especially the DISCUSSION concerning broadcast storms.

The packet send interface between the IP and link layers MUST include the 5-bit TOS field (see Section 3.2.1.6).

The link layer MUST NOT report a Destination Unreachable error to IP solely because there is no ARP cache entry for a destination.

2.5 LINK LAYER REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Trailer encapsulation	2.3.1			x			
Send Trailers by default without negotiation	2.3.1					x	
ARP	2.3.2						
Flush out-of-date ARP cache entries	2.3.2.1	x					
Prevent ARP floods	2.3.2.1	x					
Cache timeout configurable	2.3.2.1		x				
Save at least one (latest) unresolved pkt	2.3.2.2		x				
Ethernet and IEEE 802 Encapsulation	2.3.3						
Host able to:	2.3.3						
Send & receive RFC-894 encapsulation	2.3.3	x					
Receive RFC-1042 encapsulation	2.3.3		x				
Send RFC-1042 encapsulation	2.3.3			x			
Then config. sw. to select, RFC-894 dflt	2.3.3	x					
Send K1=6 encapsulation	2.3.3					x	
Use ARP on Ethernet and IEEE 802 nets	2.3.3	x					
Link layer report b'casts to IP layer	2.4	x					
IP layer pass TOS to link layer	2.4	x					
No ARP cache entry treated as Dest. Unreach.	2.4					x	

3. INTERNET LAYER PROTOCOLS

3.1 INTRODUCTION

The Robustness Principle: "Be liberal in what you accept, and conservative in what you send" is particularly important in the Internet layer, where one misbehaving host can deny Internet service to many other hosts.

The protocol standards used in the Internet layer are:

- RFC-791 [IP:1] defines the IP protocol and gives an introduction to the architecture of the Internet.
- RFC-792 [IP:2] defines ICMP, which provides routing, diagnostic and error functionality for IP. Although ICMP messages are encapsulated within IP datagrams, ICMP processing is considered to be (and is typically implemented as) part of the IP layer. See Section 3.2.2.
- RFC-950 [IP:3] defines the mandatory subnet extension to the addressing architecture.
- RFC-1112 [IP:4] defines the Internet Group Management Protocol IGMP, as part of a recommended extension to hosts and to the host-gateway interface to support Internet-wide multicasting at the IP level. See Section 3.2.3.

  The target of an IP multicast may be an arbitrary group of Internet hosts. IP multicasting is designed as a natural extension of the link-layer multicasting facilities of some networks, and it provides a standard means for local access to such link-layer multicasting facilities.

Other important references are listed in Section 5 of this document.

The Internet layer of host software MUST implement both IP and ICMP. See Section 3.3.7 for the requirements on support of IGMP.

The host IP layer has two basic functions: (1) choose the "next hop" gateway or host for outgoing IP datagrams and (2) reassemble incoming IP datagrams. The IP layer may also (3) implement intentional fragmentation of outgoing datagrams. Finally, the IP layer must (4) provide diagnostic and error functionality. We expect that IP layer functions may increase somewhat in the future, as further Internet control and management facilities are developed.

For normal datagrams, the processing is straightforward. For incoming datagrams, the IP layer:

(1) verifies that the datagram is correctly formatted;

(2) verifies that it is destined to the local host;

(3) processes options;

(4) reassembles the datagram if necessary; and

(5) passes the encapsulated message to the appropriate transport-layer protocol module.

For outgoing datagrams, the IP layer:

(1) sets any fields not set by the transport layer;

(2) selects the correct first hop on the connected network (a process called "routing");

(3) fragments the datagram if necessary and if intentional fragmentation is implemented (see Section 3.3.3); and

(4) passes the packet(s) to the appropriate link-layer driver.

A host is said to be multihomed if it has multiple IP addresses. Multihoming introduces considerable confusion and complexity into the protocol suite, and it is an area in which the Internet architecture falls seriously short of solving all problems. There are two distinct problem areas in multihoming:

(1) Local multihoming -- the host itself is multihomed; or

(2) Remote multihoming -- the local host needs to communicate with a remote multihomed host.

At present, remote multihoming MUST be handled at the application layer, as discussed in the companion RFC [INTRO:1]. A host MAY support local multihoming, which is discussed in this document, and in particular in Section 3.3.4.

Any host that forwards datagrams generated by another host is acting as a gateway and MUST also meet the specifications laid out in the gateway requirements RFC [INTRO:2]. An Internet host that includes embedded gateway code MUST have a configuration switch to disable the gateway function, and this switch MUST default to the non-gateway mode. In this mode, a datagram arriving through one interface will not be forwarded to another host or gateway (unless it is source-routed), regardless of whether the host is single-homed or multihomed. The host software MUST NOT automatically move into gateway mode if the host has more than one interface, as the operator of the machine may neither want to provide that service nor be competent to do so.

In the following, the action specified in certain cases is to "silently discard" a received datagram. This means that the datagram will be discarded without further processing and that the host will not send any ICMP error message (see Section 3.2.2) as a result. However, for diagnosis of problems a host SHOULD provide the capability of logging the error (see Section 1.2.3), including the contents of the silently-discarded datagram, and SHOULD record the event in a statistics counter.

DISCUSSION:

Silent discard of erroneous datagrams is generally intended to prevent "broadcast storms".

3.2 PROTOCOL WALK-THROUGH

3.2.1 Internet Protocol -- IP

3.2.1.1 Version Number: RFC-791 Section 3.1

A datagram whose version number is not 4 MUST be silently discarded.

3.2.1.2 Checksum: RFC-791 Section 3.1

A host MUST verify the IP header checksum on every received datagram and silently discard every datagram that has a bad checksum.

3.2.1.3 Addressing: RFC-791 Section 3.2

There are now five classes of IP addresses: Class A through Class E. Class D addresses are used for IP multicasting [IP:4], while Class E addresses are reserved for experimental use.

A multicast (Class D) address is a 28-bit logical address that stands for a group of hosts, and may be either permanent or transient. Permanent multicast addresses are allocated by the Internet Assigned Number Authority [INTRO:6], while transient addresses may be allocated dynamically to transient groups. Group membership is determined dynamically using IGMP [IP:4].

We now summarize the important special cases for Class A, B, and C IP addresses, using the following notation for an IP address:

{ <Network-number>, <Host-number> }

or

{ <Network-number>, <Subnet-number>, <Host-number> }

and the notation "-1" for a field that contains all 1 bits. This notation is not intended to imply that the 1-bits in an address mask need be contiguous.

(a) { 0, 0 }

This host on this network. MUST NOT be sent, except as a source address as part of an initialization procedure by which the host learns its own IP address.

See also Section 3.3.6 for a non-standard use of {0,0}.

(b) { 0, <Host-number> }

Specified host on this network. It MUST NOT be sent, except as a source address as part of an initialization procedure by which the host learns its full IP address.

(c) { -1, -1 }

Limited broadcast. It MUST NOT be used as a source address.

A datagram with this destination address will be received by every host on the connected physical network but will not be forwarded outside that network.

(d) { <Network-number>, -1 }

Directed broadcast to the specified network. It MUST NOT be used as a source address.

(e) { <Network-number>, <Subnet-number>, -1 }

Directed broadcast to the specified subnet. It MUST NOT be used as a source address.

(f) { <Network-number>, -1, -1 }

Directed broadcast to all subnets of the specified subnetted network. It MUST NOT be used as a source address.

(g) { 127, <any> }

Internal host loopback address. Addresses of this form MUST NOT appear outside a host.

The <Network-number> is administratively assigned so that its value will be unique in the entire world.

IP addresses are not permitted to have the value 0 or -1 for any of the <Host-number>, <Network-number>, or <Subnet-number> fields (except in the special cases listed above). This implies that each of these fields will be at least two bits long.

For further discussion of broadcast addresses, see Section 3.3.6.

A host MUST support the subnet extensions to IP [IP:3]. As a result, there will be an address mask of the form: {-1, -1, 0} associated with each of the host's local IP addresses; see Sections 3.2.2.9 and 3.3.1.1.

When a host sends any datagram, the IP source address MUST be one of its own IP addresses (but not a broadcast or multicast address).

A host MUST silently discard an incoming datagram that is not destined for the host. An incoming datagram is destined for the host if the datagram's destination address field is:

(1) (one of) the host's IP address(es); or

(2) an IP broadcast address valid for the connected network; or

(3) the address for a multicast group of which the host is a member on the incoming physical interface.

For most purposes, a datagram addressed to a broadcast or multicast destination is processed as if it had been addressed to one of the host's IP addresses; we use the term "specific-destination address" for the equivalent local IP address of the host. The specific-destination address is defined to be the destination address in the IP header unless the header contains a broadcast or multicast address, in which case the specific-destination is an IP address assigned to the physical interface on which the datagram arrived.

A host MUST silently discard an incoming datagram containing an IP source address that is invalid by the rules of this section. This validation could be done in either the IP layer or by each protocol in the transport layer.

DISCUSSION:

A mis-addressed datagram might be caused by a link-layer broadcast of a unicast datagram or by a gateway or host that is confused or mis-configured.

An architectural goal for Internet hosts was to allow IP addresses to be featureless 32-bit numbers, avoiding algorithms that required a knowledge of the IP address format. Otherwise, any future change in the format or interpretation of IP addresses will require host software changes. However, validation of broadcast and multicast addresses violates this goal; a few other violations are described elsewhere in this document.

Implementers should be aware that applications depending upon the all-subnets directed broadcast address (f) may be unusable on some networks. All-subnets broadcast is not widely implemented in vendor gateways at present, and even when it is implemented, a particular network administration may disable it in the gateway configuration.

3.2.1.4 Fragmentation and Reassembly: RFC-791 Section 3.2

The Internet model requires that every host support reassembly. See Sections 3.3.2 and 3.3.3 for the requirements on fragmentation and reassembly.

3.2.1.5 Identification: RFC-791 Section 3.2

When sending an identical copy of an earlier datagram, a host MAY optionally retain the same Identification field in the copy.

DISCUSSION:

Some Internet protocol experts have maintained that when a host sends an identical copy of an earlier datagram, the new copy should contain the same Identification value as the original. There are two suggested advantages: (1) if the datagrams are fragmented and some of the fragments are lost, the receiver may be able to reconstruct a complete datagram from fragments of the original and the copies; (2) a congested gateway might use the IP Identification field (and Fragment Offset) to discard duplicate datagrams from the queue.

However, the observed patterns of datagram loss in the Internet do not favor the probability of retransmitted fragments filling reassembly gaps, while other mechanisms (e.g., TCP repacketizing upon retransmission) tend to prevent retransmission of an identical datagram [IP:9]. Therefore, we believe that retransmitting the same Identification field is not useful. Also, a connectionless transport protocol like UDP would require the cooperation of the application programs to retain the same Identification value in identical datagrams.

3.2.1.6 Type-of-Service: RFC-791 Section 3.2

The "Type-of-Service" byte in the IP header is divided into two sections: the Precedence field (high-order 3 bits), and a field that is customarily called "Type-of-Service" or "TOS" (low-order 5 bits). In this document, all references to "TOS" or the "TOS field" refer to the low-order 5 bits only.

The Precedence field is intended for Department of Defense applications of the Internet protocols. The use of non-zero values in this field is outside the scope of this document and the IP standard specification. Vendors should consult the Defense Communication Agency (DCA) for guidance on the IP Precedence field and its implications for other protocol layers. However, vendors should note that the use of precedence will most likely require that its value be passed between protocol layers in just the same way as the TOS field is passed.

The IP layer MUST provide a means for the transport layer to set the TOS field of every datagram that is sent; the default is all zero bits. The IP layer SHOULD pass received

TOS values up to the transport layer.

The particular link-layer mappings of TOS contained in RFC-795 SHOULD NOT be implemented.

DISCUSSION:
While the TOS field has been little used in the past, it is expected to play an increasing role in the near future. The TOS field is expected to be used to control two aspects of gateway operations: routing and queueing algorithms. See Section 2 of [INTRO:1] for the requirements on application programs to specify TOS values.

The TOS field may also be mapped into link-layer service selectors. This has been applied to provide effective sharing of serial lines by different classes of TCP traffic, for example. However, the mappings suggested in RFC-795 for networks that were included in the Internet as of 1981 are now obsolete.

3.2.1.7 Time-to-Live: RFC-791 Section 3.2

A host MUST NOT send a datagram with a Time-to-Live (TTL) value of zero.

A host MUST NOT discard a datagram just because it was received with TTL less than 2.

The IP layer MUST provide a means for the transport layer to set the TTL field of every datagram that is sent. When a fixed TTL value is used, it MUST be configurable. The current suggested value will be published in the "Assigned Numbers" RFC.

DISCUSSION:
The TTL field has two functions: limit the lifetime of TCP segments (see RFC-793 [TCP:1], p. 28), and terminate Internet routing loops. Although TTL is a time in seconds, it also has some attributes of a hop-count, since each gateway is required to reduce the TTL field by at least one.

The intent is that TTL expiration will cause a datagram to be discarded by a gateway but not by the destination host; however, hosts that act as gateways by forwarding datagrams must follow the gateway rules for TTL.

A higher-layer protocol may want to set the TTL in order to implement an "expanding scope" search for some Internet resource. This is used by some diagnostic tools, and is expected to be useful for locating the "nearest" server of a given class using IP multicasting, for example. A particular transport protocol may also want to specify its own TTL bound on maximum datagram lifetime.

A fixed value must be at least big enough for the Internet "diameter," i.e., the longest possible path. A reasonable value is about twice the diameter, to allow for continued Internet growth.

3.2.1.8 Options: RFC-791 Section 3.2

There MUST be a means for the transport layer to specify IP options to be included in transmitted IP datagrams (see Section 3.4).

All IP options (except NOP or END-OF-LIST) received in datagrams MUST be passed to the transport layer (or to ICMP processing when the datagram is an ICMP message). The IP and transport layer MUST each interpret those IP options that they understand and silently ignore the others.

Later sections of this document discuss specific IP option support required by each of ICMP, TCP, and UDP.

DISCUSSION:

Passing all received IP options to the transport layer is a deliberate "violation of strict layering" that is designed to ease the introduction of new transport-relevant IP options in the future. Each layer must pick out any options that are relevant to its own processing and ignore the rest. For this purpose, every IP option except NOP and END-OF-LIST will include a specification of its own length.

This document does not define the order in which a receiver must process multiple options in the same IP header. Hosts sending multiple options must be aware that this introduces an ambiguity in the meaning of certain options when combined with a source-route option.

IMPLEMENTATION:

The IP layer must not crash as the result of an option length that is outside the possible range. For example, erroneous option lengths have been observed to put some IP implementations into infinite loops.

Here are the requirements for specific IP options:

(a) Security Option

Some environments require the Security option in every datagram; such a requirement is outside the scope of this document and the IP standard specification. Note, however, that the security options described in RFC-791 and RFC-1038 are obsolete. For DoD applications, vendors should consult [IP:8] for guidance.

(b) Stream Identifier Option

This option is obsolete; it SHOULD NOT be sent, and it MUST be silently ignored if received.

(c) Source Route Options

A host MUST support originating a source route and MUST be able to act as the final destination of a source route.

If host receives a datagram containing a completed source route (i.e., the pointer points beyond the last field), the datagram has reached its final destination; the option as received (the recorded route) MUST be passed up to the transport layer (or to ICMP message processing). This recorded route will be reversed and used to form a return source route for reply datagrams (see discussion of IP Options in Section 4). When a return source route is built, it MUST be correctly formed even if the recorded route included the source host (see case (B) in the discussion below).

An IP header containing more than one Source Route option MUST NOT be sent; the effect on routing of multiple Source Route options is implementation-specific.

Section 3.3.5 presents the rules for a host acting as an intermediate hop in a source route, i.e., forwarding a source-routed datagram.

DISCUSSION:

If a source-routed datagram is fragmented, each fragment will contain a copy of the source route. Since the processing of IP options (including a source route) must precede reassembly, the original datagram will not be reassembled until the final destination is reached.

Suppose a source routed datagram is to be routed from host S to host D via gateways G1, G2, ... Gn. There was an ambiguity in the specification over whether the source route option in a datagram sent out by S should be (A) or (B):

(A): {>>G2, G3, ... Gn, D} <--- CORRECT

(B): {S, >>G2, G3, ... Gn, D} <---- WRONG

(where >> represents the pointer). If (A) is sent, the datagram received at D will contain the option: {G1, G2, ... Gn >>}, with S and D as the IP source and destination addresses. If (B) were sent, the datagram received at D would again contain S and D as the same IP source and destination addresses, but the option would be: {S, G1, ...Gn >>}; i.e., the originating host would be the first hop in the route.

(d) Record Route Option

Implementation of originating and processing the Record Route option is OPTIONAL.

(e) Timestamp Option

Implementation of originating and processing the Timestamp option is OPTIONAL. If it is implemented, the following rules apply:

o The originating host MUST record a timestamp in a Timestamp option whose Internet address fields are not pre-specified or whose first pre-specified address is the host's interface address.

- The destination host MUST (if possible) add the current timestamp to a Timestamp option before passing the option to the transport layer or to ICMP for processing.
- A timestamp value MUST follow the rules given in Section 3.2.2.8 for the ICMP Timestamp message.

3.2.2 Internet Control Message Protocol -- ICMP

ICMP messages are grouped into two classes.

*

ICMP error messages:

Destination Unreachable	(see Section 3.2.2.1)
Redirect	(see Section 3.2.2.2)
Source Quench	(see Section 3.2.2.3)
Time Exceeded	(see Section 3.2.2.4)
Parameter Problem	(see Section 3.2.2.5)

*

ICMP query messages:

Echo	(see Section 3.2.2.6)
Information	(see Section 3.2.2.7)
Timestamp	(see Section 3.2.2.8)
Address Mask	(see Section 3.2.2.9)

If an ICMP message of unknown type is received, it MUST be silently discarded.

Every ICMP error message includes the Internet header and at least the first 8 data octets of the datagram that triggered the error; more than 8 octets MAY be sent; this header and data MUST be unchanged from the received datagram.

In those cases where the Internet layer is required to pass an ICMP error message to the transport layer, the IP protocol number MUST be extracted from the original header and used to select the appropriate transport protocol entity to handle the error.

An ICMP error message SHOULD be sent with normal (i.e., zero) TOS bits.

An ICMP error message MUST NOT be sent as the result of receiving:

* an ICMP error message, or

* a datagram destined to an IP broadcast or IP multicast address, or

* a datagram sent as a link-layer broadcast, or

* a non-initial fragment, or

* a datagram whose source address does not define a single host -- e.g., a zero address, a loopback address, a broadcast address, a multicast address, or a Class E address.

NOTE: THESE RESTRICTIONS TAKE PRECEDENCE OVER ANY REQUIREMENT ELSEWHERE IN THIS DOCUMENT FOR SENDING ICMP ERROR MESSAGES.

DISCUSSION:
These rules will prevent the "broadcast storms" that have resulted from hosts returning ICMP error messages in response to broadcast datagrams. For example, a broadcast UDP segment to a non-existent port could trigger a flood of ICMP Destination Unreachable datagrams from all machines that do not have a client for that destination port. On a large Ethernet, the resulting collisions can render the network useless for a second or more.

Every datagram that is broadcast on the connected network should have a valid IP broadcast address as its IP destination (see Section 3.3.6). However, some hosts violate this rule. To be certain to detect broadcast datagrams, therefore, hosts are required to check for a link-layer broadcast as well as an IP-layer broadcast address.

IMPLEMENTATION:
This requires that the link layer inform the IP layer when a link-layer broadcast datagram has been received; see Section 2.4.

3.2.2.1 Destination Unreachable: RFC-792

The following additional codes are hereby defined:

6 = destination network unknown

7 = destination host unknown

8 = source host isolated

9 = communication with destination network administratively prohibited

10 = communication with destination host administratively prohibited

11 = network unreachable for type of service

12 = host unreachable for type of service

A host SHOULD generate Destination Unreachable messages with code:

2 (Protocol Unreachable), when the designated transport protocol is not supported; or

3 (Port Unreachable), when the designated transport protocol (e.g., UDP) is unable to demultiplex the datagram but has no protocol mechanism to inform the sender.

A Destination Unreachable message that is received MUST be reported to the transport layer. The transport layer SHOULD use the information appropriately; for example, see Sections 4.1.3.3, 4.2.3.9, and 4.2.4 below. A transport protocol that has its own mechanism for notifying the sender that a port is unreachable (e.g., TCP, which sends RST segments) MUST nevertheless accept an ICMP Port Unreachable for the same purpose.

A Destination Unreachable message that is received with code 0 (Net), 1 (Host), or 5 (Bad Source Route) may result from a routing transient and MUST therefore be interpreted as only a hint, not proof, that the specified destination is unreachable [IP:11]. For example, it MUST NOT be used as proof of a dead gateway (see Section 3.3.1).

3.2.2.2 Redirect: RFC-792

A host SHOULD NOT send an ICMP Redirect message; Redirects are to be sent only by gateways.

A host receiving a Redirect message MUST update its routing information accordingly. Every host MUST be prepared to accept both Host and Network Redirects and to process them as described in Section 3.3.1.2 below.

A Redirect message SHOULD be silently discarded if the new gateway address it specifies is not on the same connected (sub-) net through which the Redirect arrived [INTRO:2, Appendix A], or if the source of the Redirect is not the current first-hop gateway for the specified destination (see Section 3.3.1).

3.2.2.3 Source Quench: RFC-792

A host MAY send a Source Quench message if it is approaching, or has reached, the point at which it is forced to discard incoming datagrams due to a shortage of reassembly buffers or other resources. See Section 2.2.3 of [INTRO:2] for suggestions on when to send Source Quench.

If a Source Quench message is received, the IP layer MUST report it to the transport layer (or ICMP processing). In general, the transport or application layer SHOULD implement a mechanism to respond to Source Quench for any protocol that can send a sequence of datagrams to the same destination and which can reasonably be expected to maintain enough state information to make this feasible. See Section 4 for the handling of Source Quench by TCP and UDP.

DISCUSSION:

A Source Quench may be generated by the target host or by some gateway in the path of a datagram. The host receiving a Source Quench should throttle itself back for a period of time, then gradually increase the transmission rate again. The mechanism to respond to Source Quench may be in the transport layer (for connection-oriented protocols like TCP) or in the application layer (for protocols that are built on top of UDP).

A mechanism has been proposed [IP:14] to make the IP layer respond directly to Source Quench by controlling the rate at which datagrams are sent, however, this proposal is currently experimental and not currently recommended.

3.2.2.4 Time Exceeded: RFC-792

An incoming Time Exceeded message MUST be passed to the transport layer.

DISCUSSION:

A gateway will send a Time Exceeded Code 0 (In Transit) message when it discards a datagram due to an expired TTL field. This indicates either a gateway routing loop or too small an initial TTL value.

A host may receive a Time Exceeded Code 1 (Reassembly Timeout) message from a destination host that has timed out and discarded an incomplete datagram; see Section 3.3.2 below. In the future, receipt of this message might be part of some "MTU discovery" procedure, to discover the maximum datagram size that can be sent on the path without fragmentation.

3.2.2.5 Parameter Problem: RFC-792

A host SHOULD generate Parameter Problem messages. An incoming Parameter Problem message MUST be passed to the transport layer, and it MAY be reported to the user.

DISCUSSION:

The ICMP Parameter Problem message is sent to the source host for any problem not specifically covered by another ICMP message. Receipt of a Parameter Problem message generally indicates some local or remote implementation error.

A new variant on the Parameter Problem message is hereby defined:

Code 1 = required option is missing.

DISCUSSION:

This variant is currently in use in the military community for a missing security option.

3.2.2.6 Echo Request/Reply: RFC-792

Every host MUST implement an ICMP Echo server function that receives Echo Requests and sends corresponding Echo Replies. A host SHOULD also implement an application-layer interface for sending an Echo Request and receiving an Echo Reply, for diagnostic purposes.

An ICMP Echo Request destined to an IP broadcast or IP multicast address MAY be silently discarded.

DISCUSSION:

This neutral provision results from a passionate debate between those who feel that ICMP Echo to a broadcast address provides a valuable diagnostic capability and those who feel that misuse of this feature can too easily create packet storms.

The IP source address in an ICMP Echo Reply MUST be the same as the specific-destination address (defined in Section 3.2.1.3) of the corresponding ICMP Echo Request message.

Data received in an ICMP Echo Request MUST be entirely included in the resulting Echo Reply. However, if sending the Echo Reply requires intentional fragmentation that is not implemented, the datagram MUST be truncated to maximum transmission size (see Section 3.3.3) and sent.

Echo Reply messages MUST be passed to the ICMP user interface, unless the corresponding Echo Request originated in the IP layer.

If a Record Route and/or Time Stamp option is received in an ICMP Echo Request, this option (these options) SHOULD be updated to include the current host and included in the IP header of the Echo Reply message, without "truncation". Thus, the recorded route will be for the entire round trip.

If a Source Route option is received in an ICMP Echo Request, the return route MUST be reversed and used as a Source Route option for the Echo Reply message.

3.2.2.7 Information Request/Reply: RFC-792

A host SHOULD NOT implement these messages.

DISCUSSION:

The Information Request/Reply pair was intended to support self-configuring systems such as diskless workstations, to allow them to discover their IP network numbers at boot time. However, the RARP and BOOTP protocols provide better mechanisms for a host to discover its own IP address.

3.2.2.8 Timestamp and Timestamp Reply: RFC-792

A host MAY implement Timestamp and Timestamp Reply. If they are implemented, the following rules MUST be followed.

- The ICMP Timestamp server function returns a Timestamp Reply to every Timestamp message that is received. If this function is implemented, it SHOULD be designed for minimum variability in delay (e.g., implemented in the kernel to avoid delay in scheduling a user process).

The following cases for Timestamp are to be handled according to the corresponding rules for ICMP Echo:

- An ICMP Timestamp Request message to an IP broadcast or IP multicast address MAY be silently discarded.
- The IP source address in an ICMP Timestamp Reply MUST be the same as the specific-destination address of the corresponding Timestamp Request message.
- If a Source-route option is received in an ICMP Echo Request, the return route MUST be reversed and used as a Source Route option for the Timestamp Reply message.
- If a Record Route and/or Timestamp option is received in a Timestamp Request, this (these) option(s) SHOULD be updated to include the current host and included in the IP header of the Timestamp Reply message.
- Incoming Timestamp Reply messages MUST be passed up to the ICMP user interface.

The preferred form for a timestamp value (the "standard value") is in units of milliseconds since midnight Universal Time. However, it may be difficult to provide this value with millisecond resolution. For example, many systems use clocks that update only at line frequency, 50 or 60 times per second. Therefore, some latitude is allowed in a "standard value":

(a) A "standard value" MUST be updated at least 15 times per second (i.e., at most the six low-order bits of the value may be undefined).

(b) The accuracy of a "standard value" MUST approximate that of operator-set CPU clocks, i.e., correct within a few minutes.

3.2.2.9 Address Mask Request/Reply: RFC-950

A host MUST support the first, and MAY implement all three, of the following methods for determining the address mask(s) corresponding to its IP address(es):

(1) static configuration information;

(2) obtaining the address mask(s) dynamically as a side-effect of the system initialization process (see [INTRO:1]); and

(3) sending ICMP Address Mask Request(s) and receiving ICMP Address Mask Reply(s).

The choice of method to be used in a particular host MUST be configurable.

When method (3), the use of Address Mask messages, is enabled, then:

(a) When it initializes, the host MUST broadcast an Address Mask Request message on the connected network corresponding to the IP address. It MUST retransmit this message a small number of times if it does not receive an immediate Address Mask Reply.

(b) Until it has received an Address Mask Reply, the host SHOULD assume a mask appropriate for the address class of the IP address, i.e., assume that the connected network is not subnetted.

(c) The first Address Mask Reply message received MUST be used to set the address mask corresponding to the particular local IP address. This is true even if the first Address Mask Reply message is "unsolicited", in which case it will have been broadcast and may arrive after the host has ceased to retransmit Address Mask Requests. Once the mask has been set by an Address Mask Reply, later Address Mask Reply messages MUST be (silently) ignored.

Conversely, if Address Mask messages are disabled, then no ICMP Address Mask Requests will be sent, and any ICMP Address Mask Replies received for that local IP address MUST be (silently) ignored.

A host SHOULD make some reasonableness check on any address mask it installs; see IMPLEMENTATION section below.

A system MUST NOT send an Address Mask Reply unless it is an authoritative agent for address masks. An authoritative agent may be a host or a gateway, but it MUST be explicitly configured as a address mask agent. Receiving an address mask via an Address Mask Reply does not give the receiver authority and MUST NOT be used as the basis for issuing Address Mask Replies.

With a statically configured address mask, there SHOULD be an additional configuration flag that determines whether the host is to act as an authoritative agent for this mask, i.e., whether it will answer Address Mask Request messages using this mask.

If it is configured as an agent, the host MUST broadcast an Address Mask Reply for the mask on the appropriate interface when it initializes.

See "System Initialization" in [INTRO:1] for more information about the use of Address Mask Request/Reply messages.

DISCUSSION

Hosts that casually send Address Mask Replies with invalid address masks have often been a serious nuisance. To prevent this, Address Mask Replies ought to be sent only by authoritative agents that have been selected by explicit administrative action.

When an authoritative agent receives an Address Mask Request message, it will send a unicast Address Mask Reply to the source IP address. If the network part of this address is zero (see (a) and (b) in 3.2.1.3), the Reply will be broadcast.

Getting no reply to its Address Mask Request messages, a host will assume there is no agent and use an unsubnetted mask, but the agent may be only temporarily unreachable. An agent will broadcast an unsolicited Address Mask Reply whenever it initializes, in order to update the masks of all hosts that have initialized in the meantime.

IMPLEMENTATION:

The following reasonableness check on an address mask is suggested: the mask is not all 1 bits, and it is either zero or else the 8 highest-order bits are on.

3.2.3 Internet Group Management Protocol IGMP

IGMP [IP:4] is a protocol used between hosts and gateways on a single network to establish hosts' membership in particular multicast groups. The gateways use this information, in conjunction with a multicast routing protocol, to support IP multicasting across the Internet.

At this time, implementation of IGMP is OPTIONAL; see Section 3.3.7 for more information. Without IGMP, a host can still participate in multicasting local to its connected networks.

3.3 SPECIFIC ISSUES

3.3.1 Routing Outbound Datagrams

The IP layer chooses the correct next hop for each datagram it sends. If the destination is on a connected network, the datagram is sent directly to the destination host; otherwise, it has to be routed to a gateway on a connected network.

3.3.1.1 Local/Remote Decision

To decide if the destination is on a connected network, the following algorithm MUST be used [see IP:3]:

(a) The address mask (particular to a local IP address for a multihomed host) is a 32-bit mask that selects the network number and subnet number fields of the corresponding IP address.

(b) If the IP destination address bits extracted by the address mask match the IP source address bits extracted by the same mask, then the destination is on the corresponding connected network, and the datagram is to be transmitted directly to the destination host.

(c) If not, then the destination is accessible only through a gateway. Selection of a gateway is described below (3.3.1.2).

A special-case destination address is handled as follows:

* For a limited broadcast or a multicast address, simply pass the datagram to the link layer for the appropriate interface.

* For a (network or subnet) directed broadcast, the datagram can use the standard routing algorithms.

The host IP layer MUST operate correctly in a minimal network environment, and in particular, when there are no gateways. For example, if the IP layer of a host insists on finding at least one gateway to initialize, the host will be unable to operate on a single isolated broadcast net.

3.3.1.2 Gateway Selection

To efficiently route a series of datagrams to the same destination, the source host MUST keep a "route cache" of mappings to next-hop gateways. A host uses the following basic algorithm on this cache to route a datagram; this algorithm is designed to put the primary routing burden on the gateways [IP:11].

(a) If the route cache contains no information for a particular destination, the host chooses a "default" gateway and sends the datagram to it. It also builds a corresponding Route Cache entry.

(b) If that gateway is not the best next hop to the destination, the gateway will forward the datagram to the best next-hop gateway and return an ICMP Redirect message to the source host.

(c) When it receives a Redirect, the host updates the next-hop gateway in the appropriate route cache entry, so later datagrams to the same destination will go directly to the best gateway.

Since the subnet mask appropriate to the destination address is generally not known, a Network Redirect message SHOULD be treated identically to a Host Redirect message; i.e., the cache entry for the destination host (only) would be updated (or created, if an entry for that host did not exist) for the new gateway.

DISCUSSION:

This recommendation is to protect against gateways that erroneously send Network Redirects for a subnetted network, in violation of the gateway requirements [INTRO:2].

When there is no route cache entry for the destination host address (and the destination is not on the connected network), the IP layer MUST pick a gateway from its list of "default" gateways. The IP layer MUST support multiple default gateways.

As an extra feature, a host IP layer MAY implement a table of "static routes". Each such static route MAY include a flag specifying whether it may be overridden by ICMP Redirects.

DISCUSSION:

A host generally needs to know at least one default gateway to get started. This information can be obtained from a configuration file or else from the host startup sequence, e.g., the BOOTP protocol (see [INTRO:1]).

It has been suggested that a host can augment its list of default gateways by recording any new gateways it learns about. For example, it can record every gateway to which it is ever redirected. Such a feature, while possibly useful in some circumstances, may cause problems in other cases (e.g., gateways are not all equal), and it is not recommended.

A static route is typically a particular preset mapping from destination host or network into a particular next-hop gateway; it might also depend on the Type-of-Service (see next section). Static routes would be set up by system administrators to override the normal automatic routing mechanism, to handle exceptional situations. However, any static routing information is a potential source of failure as configurations change or equipment fails.

3.3.1.3 Route Cache

Each route cache entry needs to include the following fields:

(1) Local IP address (for a multihomed host)

(2) Destination IP address

(3) Type(s)-of-Service

(4) Next-hop gateway IP address

Field (2) MAY be the full IP address of the destination host, or only the destination network number. Field (3), the TOS, SHOULD be included.

See Section 3.3.4.2 for a discussion of the implications of multihoming for the lookup procedure in this cache.

DISCUSSION:

Including the Type-of-Service field in the route cache and considering it in the host route algorithm will provide the necessary mechanism for the future when Type-of-Service routing is commonly used in the Internet. See Section 3.2.1.6.

Each route cache entry defines the endpoints of an Internet path. Although the connecting path may change dynamically in an arbitrary way, the transmission characteristics of the path tend to remain approximately constant over a time period longer than a single typical host-host transport connection. Therefore, a route cache entry is a natural place to cache data on the properties of the path. Examples of such properties might be the maximum unfragmented datagram size (see Section 3.3.3), or the average round-trip delay measured by a transport protocol. This data will generally be both gathered and used by a higher layer protocol, e.g., by TCP, or by an application using UDP. Experiments are currently in progress on caching path properties in this manner.

There is no consensus on whether the route cache should be keyed on destination host addresses alone, or allow both host and network addresses. Those who favor the use of only host addresses argue that:

(1) As required in Section 3.3.1.2, Redirect messages will generally result in entries keyed on destination host addresses; the simplest and most general scheme would be to use host addresses always.

(2) The IP layer may not always know the address mask for a network address in a complex subnetted environment.

(3) The use of only host addresses allows the destination address to be used as a pure 32-bit number, which may allow the Internet architecture to be more easily extended in the future without

any change to the hosts.

The opposing view is that allowing a mixture of destination hosts and networks in the route cache:

(1) Saves memory space.

(2) Leads to a simpler data structure, easily combining the cache with the tables of default and static routes (see below).

(3) Provides a more useful place to cache path properties, as discussed earlier.

IMPLEMENTATION:
The cache needs to be large enough to include entries for the maximum number of destination hosts that may be in use at one time.

A route cache entry may also include control information used to choose an entry for replacement. This might take the form of a "recently used" bit, a use count, or a last-used timestamp, for example. It is recommended that it include the time of last modification of the entry, for diagnostic purposes.

An implementation may wish to reduce the overhead of scanning the route cache for every datagram to be transmitted. This may be accomplished with a hash table to speed the lookup, or by giving a connection-oriented transport protocol a "hint" or temporary handle on the appropriate cache entry, to be passed to the IP layer with each subsequent datagram.

Although we have described the route cache, the lists of default gateways, and a table of static routes as conceptually distinct, in practice they may be combined into a single "routing table" data structure.

3.3.1.4 Dead Gateway Detection

The IP layer MUST be able to detect the failure of a "next-hop" gateway that is listed in its route cache and to choose an alternate gateway (see Section 3.3.1.5).

Dead gateway detection is covered in some detail in RFC-816 [IP:11]. Experience to date has not produced a complete algorithm which is totally satisfactory, though it has identified several forbidden paths and promising techniques.

- A particular gateway SHOULD NOT be used indefinitely in the absence of positive indications that it is functioning.

- Active probes such as "pinging" (i.e., using an ICMP Echo Request/Reply exchange) are expensive and scale poorly. In particular, hosts MUST NOT actively check the status of a first-hop gateway by simply pinging the gateway continuously.

- Even when it is the only effective way to verify a gateway's status, pinging MUST be used only when traffic is being sent to the gateway and when there is no other positive indication to suggest that the gateway is functioning.

- To avoid pinging, the layers above and/or below the Internet layer SHOULD be able to give "advice" on the status of route cache entries when either positive (gateway OK) or negative (gateway dead) information is available.

DISCUSSION:

If an implementation does not include an adequate mechanism for detecting a dead gateway and re-routing, a gateway failure may cause datagrams to apparently vanish into a "black hole". This failure can be extremely confusing for users and difficult for network personnel to debug.

The dead-gateway detection mechanism must not cause unacceptable load on the host, on connected networks, or on first-hop gateway(s). The exact constraints on the timeliness of dead gateway detection and on acceptable load may vary somewhat depending on the nature of the host's mission, but a host generally needs to detect a failed first-hop gateway quickly enough that transport-layer connections will not break before an alternate gateway can be selected.

Passing advice from other layers of the protocol stack complicates the interfaces between the layers, but it is the preferred approach to dead gateway detection. Advice can come from almost any part of the IP/TCP architecture, but it is expected to come primarily from the transport and link layers. Here are some possible sources for gateway advice:

- o TCP or any connection-oriented transport protocol should be able to give negative advice, e.g., triggered by excessive retransmissions.
- o TCP may give positive advice when (new) data is acknowledged. Even though the route may be asymmetric, an ACK for new data proves that the acknowleged data must have been transmitted successfully.
- o An ICMP Redirect message from a particular gateway should be used as positive advice about that gateway.
- o Link-layer information that reliably detects and reports host failures (e.g., ARPANET Destination Dead messages) should be used as negative advice.
- o Failure to ARP or to re-validate ARP mappings may be used as negative advice for the corresponding IP address.
- o Packets arriving from a particular link-layer address are evidence that the system at this address is alive. However, turning this information into advice about gateways requires mapping the link-layer address into an IP address, and then checking that IP address against the gateways pointed to by the route cache. This is probably prohibitively inefficient.

Note that positive advice that is given for every datagram received may cause unacceptable overhead in the implementation.

While advice might be passed using required arguments in all interfaces to the IP layer, some transport and application layer protocols cannot deduce the correct advice. These interfaces must therefore allow a neutral value for advice, since either always-positive or always-negative advice leads to incorrect behavior.

There is another technique for dead gateway detection that has been commonly used but is not recommended.

This technique depends upon the host passively receiving ("wiretapping") the Interior Gateway Protocol (IGP) datagrams that the gateways are broadcasting to each other. This approach has the drawback that a host needs to recognize all the interior gateway protocols that gateways may use (see [INTRO:2]). In addition, it only works on a broadcast network.

At present, pinging (i.e., using ICMP Echo messages) is the mechanism for gateway probing when absolutely required. A successful ping guarantees that the addressed interface and its associated machine are up, but it does not guarantee that the machine is a gateway as opposed to a host. The normal inference is that if a Redirect or other evidence indicates that a machine was a gateway, successful pings will indicate that the machine is still up and hence still a gateway. However, since a host silently discards packets that a gateway would forward or redirect, this assumption could sometimes fail. To avoid this problem, a new ICMP message under development will ask "are you a gateway?"

IMPLEMENTATION:
The following specific algorithm has been suggested:

- Associate a "reroute timer" with each gateway pointed to by the route cache. Initialize the timer to a value Tr, which must be small enough to allow detection of a dead gateway before transport connections time out.
- Positive advice would reset the reroute timer to Tr. Negative advice would reduce or zero the reroute timer.
- Whenever the IP layer used a particular gateway to route a datagram, it would check the corresponding reroute timer. If the timer had expired (reached zero), the IP layer would send a ping to the gateway, followed immediately by the datagram.
- The ping (ICMP Echo) would be sent again if necessary, up to N times. If no ping reply was received in N tries, the gateway would be assumed to have failed, and a new first-hop gateway would be chosen for all cache entries pointing to the failed gateway.

Note that the size of Tr is inversely related to the amount of advice available. Tr should be large enough to insure that:

- Any pinging will be at a low level (e.g., <10%) of all packets sent to a gateway from the host, AND
- pinging is infrequent (e.g., every 3 minutes)

Since the recommended algorithm is concerned with the gateways pointed to by route cache entries, rather than the cache entries themselves, a two level data structure (perhaps coordinated with ARP or similar caches) may be desirable for implementing a route cache.

3.3.1.5 New Gateway Selection

If the failed gateway is not the current default, the IP layer can immediately switch to a default gateway. If it is the current default that failed, the IP layer MUST select a different default gateway (assuming more than one default is known) for the failed route and for establishing new routes.

DISCUSSION:
When a gateway does fail, the other gateways on the connected network will learn of the failure through some inter-gateway routing protocol. However, this will not happen instantaneously, since gateway routing protocols typically have a settling time of 30-60 seconds. If the host switches to an alternative gateway before the gateways have agreed on the failure, the new target gateway will probably forward the datagram to the failed gateway and send a Redirect back to the host pointing to the failed gateway (!). The result is likely to be a rapid oscillation in the contents of the host's route cache during the gateway settling period. It has been proposed that the dead-gateway logic should include some hysteresis mechanism to prevent such oscillations. However, experience has not shown any harm from such oscillations, since service cannot be restored to the host until the gateways' routing information does settle down.

IMPLEMENTATION:
One implementation technique for choosing a new default gateway is to simply round-robin among the default gateways in the host's list. Another is to rank the

gateways in priority order, and when the current default gateway is not the highest priority one, to "ping" the higher-priority gateways slowly to detect when they return to service. This pinging can be at a very low rate, e.g., 0.005 per second.

3.3.1.6 Initialization

The following information MUST be configurable:

(1) IP address(es).

(2) Address mask(s).

(3) A list of default gateways, with a preference level.

A manual method of entering this configuration data MUST be provided. In addition, a variety of methods can be used to determine this information dynamically; see the section on "Host Initialization" in [INTRO:1].

DISCUSSION:
Some host implementations use "wiretapping" of gateway protocols on a broadcast network to learn what gateways exist. A standard method for default gateway discovery is under development.

3.3.2 Reassembly

The IP layer MUST implement reassembly of IP datagrams.

We designate the largest datagram size that can be reassembled by EMTU_R ("Effective MTU to receive"); this is sometimes called the "reassembly buffer size". EMTU_R MUST be greater than or equal to 576, SHOULD be either configurable or indefinite, and SHOULD be greater than or equal to the MTU of the connected network(s).

DISCUSSION:
A fixed EMTU_R limit should not be built into the code because some application layer protocols require EMTU_R values larger than 576.

IMPLEMENTATION:
An implementation may use a contiguous reassembly buffer for each datagram, or it may use a more complex data structure that places no definite limit on the reassembled datagram size; in the latter case, EMTU_R is said to be

"indefinite".

Logically, reassembly is performed by simply copying each fragment into the packet buffer at the proper offset. Note that fragments may overlap if successive retransmissions use different packetizing but the same reassembly Id.

The tricky part of reassembly is the bookkeeping to determine when all bytes of the datagram have been reassembled. We recommend Clark's algorithm [IP:10] that requires no additional data space for the bookkeeping. However, note that, contrary to [IP:10], the first fragment header needs to be saved for inclusion in a possible ICMP Time Exceeded (Reassembly Timeout) message.

There MUST be a mechanism by which the transport layer can learn MMS_R, the maximum message size that can be received and reassembled in an IP datagram (see GET_MAXSIZES calls in Section 3.4). If EMTU_R is not indefinite, then the value of MMS_R is given by:

    MMS_R = EMTU_R - 20

since 20 is the minimum size of an IP header.

There MUST be a reassembly timeout. The reassembly timeout value SHOULD be a fixed value, not set from the remaining TTL. It is recommended that the value lie between 60 seconds and 120 seconds. If this timeout expires, the partially-reassembled datagram MUST be discarded and an ICMP Time Exceeded message sent to the source host (if fragment zero has been received).

DISCUSSION:

The IP specification says that the reassembly timeout should be the remaining TTL from the IP header, but this does not work well because gateways generally treat TTL as a simple hop count rather than an elapsed time. If the reassembly timeout is too small, datagrams will be discarded unnecessarily, and communication may fail. The timeout needs to be at least as large as the typical maximum delay across the Internet. A realistic minimum reassembly timeout would be 60 seconds.

It has been suggested that a cache might be kept of round-trip times measured by transport protocols for various destinations, and that these values might be used to dynamically determine a reasonable reassembly timeout value. Further investigation of this approach is required.

If the reassembly timeout is set too high, buffer resources in the receiving host will be tied up too long, and the MSL (Maximum Segment Lifetime) [TCP:1] will be larger than necessary. The MSL controls the maximum rate at which fragmented datagrams can be sent using distinct values of the 16-bit Ident field; a larger MSL lowers the maximum rate. The TCP specification [TCP:1] arbitrarily assumes a value of 2 minutes for MSL. This sets an upper limit on a reasonable reassembly timeout value.

3.3.3 Fragmentation

Optionally, the IP layer MAY implement a mechanism to fragment outgoing datagrams intentionally.

We designate by EMTU_S ("Effective MTU for sending") the maximum IP datagram size that may be sent, for a particular combination of IP source and destination addresses and perhaps TOS.

A host MUST implement a mechanism to allow the transport layer to learn MMS_S, the maximum transport-layer message size that may be sent for a given {source, destination, TOS} triplet (see GET_MAXSIZES call in Section 3.4). If no local fragmentation is performed, the value of MMS_S will be:

MMS_S = EMTU_S - <IP header size>

and EMTU_S must be less than or equal to the MTU of the network interface corresponding to the source address of the datagram. Note that <IP header size> in this equation will be 20, unless the IP reserves space to insert IP options for its own purposes in addition to any options inserted by the transport layer.

A host that does not implement local fragmentation MUST ensure that the transport layer (for TCP) or the application layer (for UDP) obtains MMS_S from the IP layer and does not send a datagram exceeding MMS_S in size.

It is generally desirable to avoid local fragmentation and to choose EMTU_S low enough to avoid fragmentation in any gateway along the path. In the absence of actual knowledge of the minimum MTU along the path, the IP layer SHOULD use EMTU_S <= 576 whenever the destination address is not on a connected network, and otherwise use the connected network's

MTU.

The MTU of each physical interface MUST be configurable.

A host IP layer implementation MAY have a configuration flag "All-Subnets-MTU", indicating that the MTU of the connected network is to be used for destinations on different subnets within the same network, but not for other networks. Thus, this flag causes the network class mask, rather than the subnet address mask, to be used to choose an EMTU_S. For a multihomed host, an "All-Subnets-MTU" flag is needed for each network interface.

DISCUSSION:

Picking the correct datagram size to use when sending data is a complex topic [IP:9].

(a) In general, no host is required to accept an IP datagram larger than 576 bytes (including header and data), so a host must not send a larger datagram without explicit knowledge or prior arrangement with the destination host. Thus, MMS_S is only an upper bound on the datagram size that a transport protocol may send; even when MMS_S exceeds 556, the transport layer must limit its messages to 556 bytes in the absence of other knowledge about the destination host.

(b) Some transport protocols (e.g., TCP) provide a way to explicitly inform the sender about the largest datagram the other end can receive and reassemble [IP:7]. There is no corresponding mechanism in the IP layer.

A transport protocol that assumes an EMTU_R larger than 576 (see Section 3.3.2), can send a datagram of this larger size to another host that implements the same protocol.

(c) Hosts should ideally limit their EMTU_S for a given destination to the minimum MTU of all the networks along the path, to avoid any fragmentation. IP fragmentation, while formally correct, can create a serious transport protocol performance problem, because loss of a single fragment means all the fragments in the segment must be retransmitted [IP:9].

Since nearly all networks in the Internet currently support an MTU of 576 or greater, we strongly recommend the use of 576 for datagrams sent to non-local networks.

It has been suggested that a host could determine the MTU over a given path by sending a zero-offset datagram fragment and waiting for the receiver to time out the reassembly (which cannot complete!) and return an ICMP Time Exceeded message. This message would include the largest remaining fragment header in its body. More direct mechanisms are being experimented with, but have not yet been adopted (see e.g., RFC-1063).

3.3.4 Local Multihoming

3.3.4.1 Introduction

A multihomed host has multiple IP addresses, which we may think of as "logical interfaces". These logical interfaces may be associated with one or more physical interfaces, and these physical interfaces may be connected to the same or different networks.

Here are some important cases of multihoming:

(a) Multiple Logical Networks

The Internet architects envisioned that each physical network would have a single unique IP network (or subnet) number. However, LAN administrators have sometimes found it useful to violate this assumption, operating a LAN with multiple logical networks per physical connected network.

If a host connected to such a physical network is configured to handle traffic for each of N different logical networks, then the host will have N logical interfaces. These could share a single physical interface, or might use N physical interfaces to the same network.

(b) Multiple Logical Hosts

When a host has multiple IP addresses that all have the same <Network-number> part (and the same <Subnet-number> part, if any), the logical interfaces are known as "logical hosts". These logical interfaces might share a single physical interface or might use separate physical interfaces to the same physical network.

(c) Simple Multihoming

In this case, each logical interface is mapped into a separate physical interface and each physical interface is connected to a different physical network. The term "multihoming" was originally applied only to this case, but it is now applied more generally.

A host with embedded gateway functionality will typically fall into the simple multihoming case. Note, however, that a host may be simply multihomed without containing an embedded gateway, i.e., without forwarding datagrams from one connected network to another.

This case presents the most difficult routing problems. The choice of interface (i.e., the choice of first-hop network) may significantly affect performance or even reachability of remote parts of the Internet.

Finally, we note another possibility that is NOT multihoming: one logical interface may be bound to multiple physical interfaces, in order to increase the reliability or throughput between directly connected machines by providing alternative physical paths between them. For instance, two systems might be connected by multiple point-to-point links. We call this "link-layer multiplexing". With link-layer multiplexing, the protocols above the link layer are unaware that multiple physical interfaces are present; the link-layer device driver is responsible for multiplexing and routing packets across the physical interfaces.

In the Internet protocol architecture, a transport protocol instance ("entity") has no address of its own, but instead uses a single Internet Protocol (IP) address. This has implications for the IP, transport, and application layers, and for the interfaces between them. In particular, the application software may have to be aware of the multiple IP addresses of a multihomed host; in other cases, the choice can be made within the network software.

3.3.4.2 Multihoming Requirements

The following general rules apply to the selection of an IP source address for sending a datagram from a multihomed host.

(1) If the datagram is sent in response to a received datagram, the source address for the response SHOULD be the specific-destination address of the request. See Sections 4.1.3.5 and 4.2.3.7 and the "General Issues" section of [INTRO:1] for more specific requirements on higher layers.

Otherwise, a source address must be selected.

(2) An application MUST be able to explicitly specify the source address for initiating a connection or a request.

(3) In the absence of such a specification, the networking software MUST choose a source address. Rules for this choice are described below.

There are two key requirement issues related to multihoming:

(A) A host MAY silently discard an incoming datagram whose destination address does not correspond to the physical interface through which it is received.

(B) A host MAY restrict itself to sending (non-source-routed) IP datagrams only through the physical interface that corresponds to the IP source address of the datagrams.

DISCUSSION:

Internet host implementors have used two different conceptual models for multihoming, briefly summarized in the following discussion. This document takes no stand on which model is preferred; each seems to have a place. This ambivalence is reflected in the issues (A) and (B) being optional.

o Strong ES Model

The Strong ES (End System, i.e., host) model emphasizes the host/gateway (ES/IS) distinction, and would therefore substitute MUST for MAY in issues (A) and (B) above. It tends to model a multihomed host as a set of logical hosts within the same physical host.

With respect to (A), proponents of the Strong ES model note that automatic Internet routing mechanisms could not route a datagram to a physical interface that did not correspond to the destination address.

Under the Strong ES model, the route computation for an outgoing datagram is the mapping:

```
route(src IP addr, dest IP addr, TOS)
                                   -> gateway
```

Here the source address is included as a parameter in order to select a gateway that is directly reachable on the corresponding physical interface. Note that this model logically requires that in general there be at least one default gateway, and preferably multiple defaults, for each IP source address.

- Weak ES Model

  This view de-emphasizes the ES/IS distinction, and would therefore substitute MUST NOT for MAY in issues (A) and (B). This model may be the more natural one for hosts that wiretap gateway routing protocols, and is necessary for hosts that have embedded gateway functionality.

  The Weak ES Model may cause the Redirect mechanism to fail. If a datagram is sent out a physical interface that does not correspond to the destination address, the first-hop gateway will not realize when it needs to send a Redirect. On the other hand, if the host has embedded gateway functionality, then it has routing information without listening to Redirects.

  In the Weak ES model, the route computation for an outgoing datagram is the mapping:

  ```
  route(dest IP addr, TOS) -> gateway, interface
  ```

3.3.4.3 Choosing a Source Address

DISCUSSION:

When it sends an initial connection request (e.g., a TCP "SYN" segment) or a datagram service request (e.g., a UDP-based query), the transport layer on a multihomed host needs to know which source address to use. If the application does not specify it, the transport layer must ask the IP layer to perform the conceptual mapping:

```
GET_SRCADDR(remote IP addr, TOS)
                         -> local IP address
```

Here TOS is the Type-of-Service value (see Section 3.2.1.6), and the result is the desired source address. The following rules are suggested for implementing this mapping:

(a) If the remote Internet address lies on one of the (sub-) nets to which the host is directly connected, a corresponding source address may be chosen, unless the corresponding interface is known to be down.

(b) The route cache may be consulted, to see if there is an active route to the specified destination network through any network interface; if so, a local IP address corresponding to that interface may be chosen.

(c) The table of static routes, if any (see Section 3.3.1.2) may be similarly consulted.

(d) The default gateways may be consulted. If these gateways are assigned to different interfaces, the interface corresponding to the gateway with the highest preference may be chosen.

In the future, there may be a defined way for a multihomed host to ask the gateways on all connected networks for advice about the best network to use for a given destination.

IMPLEMENTATION:

It will be noted that this process is essentially the same as datagram routing (see Section 3.3.1), and therefore hosts may be able to combine the implementation of the two functions.

3.3.5 Source Route Forwarding

Subject to restrictions given below, a host MAY be able to act as an intermediate hop in a source route, forwarding a source-routed datagram to the next specified hop.

However, in performing this gateway-like function, the host MUST obey all the relevant rules for a gateway forwarding source-routed datagrams [INTRO:2]. This includes the following specific provisions, which override the corresponding host provisions given earlier in this document:

(A) TTL (ref. Section 3.2.1.7)

The TTL field MUST be decremented and the datagram perhaps discarded as specified for a gateway in [INTRO:2].

(B) ICMP Destination Unreachable (ref. Section 3.2.2.1)

A host MUST be able to generate Destination Unreachable messages with the following codes:

4 (Fragmentation Required but DF Set) when a source-routed datagram cannot be fragmented to fit into the target network;

5 (Source Route Failed) when a source-routed datagram cannot be forwarded, e.g., because of a routing problem or because the next hop of a strict source route is not on a connected network.

(C) IP Source Address (ref. Section 3.2.1.3)

A source-routed datagram being forwarded MAY (and normally will) have a source address that is not one of the IP addresses of the forwarding host.

(D) Record Route Option (ref. Section 3.2.1.8d)

A host that is forwarding a source-routed datagram containing a Record Route option MUST update that option, if it has room.

(E) Timestamp Option (ref. Section 3.2.1.8e)

A host that is forwarding a source-routed datagram containing a Timestamp Option MUST add the current timestamp to that option, according to the rules for this option.

To define the rules restricting host forwarding of source-routed datagrams, we use the term "local source-routing" if the next hop will be through the same physical interface through which the datagram arrived; otherwise, it is "non-local source-routing".

o A host is permitted to perform local source-routing without restriction.

o A host that supports non-local source-routing MUST have a configurable switch to disable forwarding, and this switch MUST default to disabled.

o The host MUST satisfy all gateway requirements for configurable policy filters [INTRO:2] restricting non-local forwarding.

If a host receives a datagram with an incomplete source route but does not forward it for some reason, the host SHOULD return an ICMP Destination Unreachable (code 5, Source Route Failed) message, unless the datagram was itself an ICMP error message.

3.3.6 Broadcasts

Section 3.2.1.3 defined the four standard IP broadcast address forms:

Limited Broadcast: {-1, -1}

Directed Broadcast: {<Network-number>,-1}

Subnet Directed Broadcast:
{<Network-number>,<Subnet-number>,-1}

All-Subnets Directed Broadcast: {<Network-number>,-1,-1}

A host MUST recognize any of these forms in the destination address of an incoming datagram.

There is a class of hosts* that use non-standard broadcast address forms, substituting 0 for -1. All hosts SHOULD recognize and accept any of these non-standard broadcast addresses as the destination address of an incoming datagram. A host MAY optionally have a configuration option to choose the 0 or the -1 form of broadcast address, for each physical interface, but this option SHOULD default to the standard (-1) form.

---

*4.2BSD Unix and its derivatives, but not 4.3BSD.

When a host sends a datagram to a link-layer broadcast address, the IP destination address MUST be a legal IP broadcast or IP multicast address.

A host SHOULD silently discard a datagram that is received via a link-layer broadcast (see Section 2.4) but does not specify an IP multicast or broadcast destination address.

Hosts SHOULD use the Limited Broadcast address to broadcast to a connected network.

DISCUSSION:

Using the Limited Broadcast address instead of a Directed Broadcast address may improve system robustness. Problems are often caused by machines that do not understand the plethora of broadcast addresses (see Section 3.2.1.3), or that may have different ideas about which broadcast addresses are in use. The prime example of the latter is machines that do not understand subnetting but are attached to a subnetted net. Sending a Subnet Broadcast for the connected network will confuse those machines, which will see it as a message to some other host.

There has been discussion on whether a datagram addressed to the Limited Broadcast address ought to be sent from all the interfaces of a multihomed host. This specification takes no stand on the issue.

3.3.7 IP Multicasting

A host SHOULD support local IP multicasting on all connected networks for which a mapping from Class D IP addresses to link-layer addresses has been specified (see below). Support for local IP multicasting includes sending multicast datagrams, joining multicast groups and receiving multicast datagrams, and leaving multicast groups. This implies support for all of [IP:4] except the IGMP protocol itself, which is OPTIONAL.

DISCUSSION:

IGMP provides gateways that are capable of multicast routing with the information required to support IP multicasting across multiple networks. At this time, multicast-routing gateways are in the experimental stage and are not widely available. For hosts that are not connected to networks with multicast-routing gateways or that do not need to receive multicast datagrams originating on other networks, IGMP serves no purpose and is therefore optional for now. However, the rest of [IP:4] is currently recommended for the purpose of providing IP-layer access to local network multicast addressing, as a preferable alternative to local broadcast addressing. It is expected that IGMP will become recommended at some future date, when multicast-routing gateways have become more widely available.

If IGMP is not implemented, a host SHOULD still join the "all-hosts" group (224.0.0.1) when the IP layer is initialized and remain a member for as long as the IP layer is active.

DISCUSSION:

Joining the "all-hosts" group will support strictly local uses of multicasting, e.g., a gateway discovery protocol, even if IGMP is not implemented.

The mapping of IP Class D addresses to local addresses is currently specified for the following types of networks:

- Ethernet/IEEE 802.3, as defined in [IP:4].
- Any network that supports broadcast but not multicast, addressing: all IP Class D addresses map to the local broadcast address.
- Any type of point-to-point link (e.g., SLIP or HDLC links): no mapping required. All IP multicast datagrams are sent as-is, inside the local framing.

Mappings for other types of networks will be specified in the future.

A host SHOULD provide a way for higher-layer protocols or applications to determine which of the host's connected network(s) support IP multicast addressing.

3.3.8 Error Reporting

Wherever practical, hosts MUST return ICMP error datagrams on detection of an error, except in those cases where returning an ICMP error message is specifically prohibited.

DISCUSSION:

A common phenomenon in datagram networks is the "black hole disease": datagrams are sent out, but nothing comes back. Without any error datagrams, it is difficult for the user to figure out what the problem is.

3.4 INTERNET/TRANSPORT LAYER INTERFACE

The interface between the IP layer and the transport layer MUST provide full access to all the mechanisms of the IP layer, including options, Type-of-Service, and Time-to-Live. The transport layer MUST either have mechanisms to set these interface parameters, or provide a path to pass them through from an application, or both.

DISCUSSION:

Applications are urged to make use of these mechanisms where applicable, even when the mechanisms are not currently effective in the Internet (e.g., TOS). This will allow these mechanisms to be immediately useful when they do become effective, without a large amount of retrofitting of host software.

We now describe a conceptual interface between the transport layer and the IP layer, as a set of procedure calls. This is an extension of the information in Section 3.3 of RFC-791 [IP:1].

- Send Datagram

```
SEND(src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt
    => result )
```

where the parameters are defined in RFC-791. Passing an Id parameter is optional; see Section 3.2.1.5.

- Receive Datagram

```
RECV(BufPTR, prot
    => result, src, dst, SpecDest, TOS, len, opt)
```

All the parameters are defined in RFC-791, except for:

SpecDest = specific-destination address of datagram
(defined in Section 3.2.1.3)

The result parameter dst contains the datagram's destination address. Since this may be a broadcast or multicast address, the SpecDest parameter (not shown in RFC-791) MUST be passed. The parameter opt contains all the IP options received in the datagram; these MUST also be passed to the transport layer.

* Select Source Address

```
GET_SRCADDR(remote, TOS)  -> local
```

remote = remote IP address
TOS = Type-of-Service
local = local IP address

See Section 3.3.4.3.

* Find Maximum Datagram Sizes

```
GET_MAXSIZES(local, remote, TOS) -> MMS_R, MMS_S
```

MMS_R = maximum receive transport-message size.
MMS_S = maximum send transport-message size.
(local, remote, TOS defined above)

See Sections 3.3.2 and 3.3.3.

* Advice on Delivery Success

```
ADVISE_DELIVPROB(sense, local, remote, TOS)
```

Here the parameter sense is a 1-bit flag indicating whether positive or negative advice is being given; see the discussion in Section 3.3.1.4. The other parameters were defined earlier.

* Send ICMP Message

```
SEND_ICMP(src, dst, TOS, TTL, BufPTR, len, Id, DF, opt)
    -> result
```

(Parameters defined in RFC-791).

Passing an Id parameter is optional; see Section 3.2.1.5. The transport layer MUST be able to send certain ICMP messages: Port Unreachable or any of the query-type messages. This function could be considered to be a special case of the SEND() call, of course; we describe it separately for clarity.

- Receive ICMP Message

  RECV_ICMP(BufPTR) -> result, src, dst, len, opt

  (Parameters defined in RFC-791).

  The IP layer MUST pass certain ICMP messages up to the appropriate transport-layer routine. This function could be considered to be a special case of the RECV() call, of course; we describe it separately for clarity.

  For an ICMP error message, the data that is passed up MUST include the original Internet header plus all the octets of the original message that are included in the ICMP message. This data will be used by the transport layer to locate the connection state information, if any.

  In particular, the following ICMP messages are to be passed up:

  o Destination Unreachable

  o Source Quench

  o Echo Reply (to ICMP user interface, unless the Echo Request originated in the IP layer)

  o Timestamp Reply (to ICMP user interface)

  o Time Exceeded

DISCUSSION:
In the future, there may be additions to this interface to pass path data (see Section 3.3.1.3) between the IP and transport layers.

3.5 INTERNET LAYER REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Implement IP and ICMP	3.1	x					
Handle remote multihoming in application layer	3.1	x					
Support local multihoming	3.1			x			
Meet gateway specs if forward datagrams	3.1	x					
Configuration switch for embedded gateway	3.1	x					1
Config switch default to non-gateway	3.1	x					1
Auto-config based on number of interfaces	3.1					x	1
Able to log discarded datagrams	3.1		x				
Record in counter	3.1		x				
Silently discard Version != 4	3.2.1.1	x					
Verify IP checksum, silently discard bad dgram	3.2.1.2	x					
Addressing:							
Subnet addressing (RFC-950)	3.2.1.3	x					
Src address must be host's own IP address	3.2.1.3	x					
Silently discard datagram with bad dest addr	3.2.1.3	x					
Silently discard datagram with bad src addr	3.2.1.3	x					
Support reassembly	3.2.1.4	x					
Retain same Id field in identical datagram	3.2.1.5			x			
TOS:							
Allow transport layer to set TOS	3.2.1.6	x					
Pass received TOS up to transport layer	3.2.1.6		x				
Use RFC-795 link-layer mappings for TOS	3.2.1.6				x		
TTL:							
Send packet with TTL of 0	3.2.1.7					x	
Discard received packets with TTL < 2	3.2.1.7					x	
Allow transport layer to set TTL	3.2.1.7	x					
Fixed TTL is configurable	3.2.1.7	x					
IP Options:							
Allow transport layer to send IP options	3.2.1.8	x					
Pass all IP options rcvd to higher layer	3.2.1.8	x					

IP layer silently ignore unknown options	3.2.1.8	x				
Security option	3.2.1.8a			x		
Send Stream Identifier option	3.2.1.8b				x	
Silently ignore Stream Identifer option	3.2.1.8b	x				
Record Route option	3.2.1.8d			x		
Timestamp option	3.2.1.8e			x		
Source Route Option:						
Originate & terminate Source Route options	3.2.1.8c	x				
Datagram with completed SR passed up to TL	3.2.1.8c	x				
Build correct (non-redundant) return route	3.2.1.8c	x				
Send multiple SR options in one header	3.2.1.8c					x
ICMP:						
Silently discard ICMP msg with unknown type	3.2.2	x				
Include more than 8 octets of orig datagram	3.2.2			x		
Included octets same as received	3.2.2	x				
Demux ICMP Error to transport protocol	3.2.2	x				
Send ICMP error message with TOS=0	3.2.2		x			
Send ICMP error message for:						
- ICMP error msg	3.2.2					x
- IP b'cast or IP m'cast	3.2.2					x
- Link-layer b'cast	3.2.2					x
- Non-initial fragment	3.2.2					x
- Datagram with non-unique src address	3.2.2					x
Return ICMP error msgs (when not prohibited)	3.3.8	x				
Dest Unreachable:						
Generate Dest Unreachable (code 2/3)	3.2.2.1		x			
Pass ICMP Dest Unreachable to higher layer	3.2.2.1	x				
Higher layer act on Dest Unreach	3.2.2.1		x			
Interpret Dest Unreach as only hint	3.2.2.1	x				
Redirect:						
Host send Redirect	3.2.2.2				x	
Update route cache when recv Redirect	3.2.2.2	x				
Handle both Host and Net Redirects	3.2.2.2	x				
Discard illegal Redirect	3.2.2.2		x			
Source Quench:						
Send Source Quench if buffering exceeded	3.2.2.3			x		
Pass Source Quench to higher layer	3.2.2.3	x				
Higher layer act on Source Quench	3.2.2.3		x			
Time Exceeded: pass to higher layer	3.2.2.4	x				
Parameter Problem:						
Send Parameter Problem messages	3.2.2.5		x			
Pass Parameter Problem to higher layer	3.2.2.5	x				
Report Parameter Problem to user	3.2.2.5			x		
ICMP Echo Request or Reply:						
Echo server and Echo client	3.2.2.6	x				

Echo client	3.2.2.6		x				
Discard Echo Request to broadcast address	3.2.2.6			x			
Discard Echo Request to multicast address	3.2.2.6			x			
Use specific-dest addr as Echo Reply src	3.2.2.6	x					
Send same data in Echo Reply	3.2.2.6	x					
Pass Echo Reply to higher layer	3.2.2.6	x					
Reflect Record Route, Time Stamp options	3.2.2.6		x				
Reverse and reflect Source Route option	3.2.2.6	x					
ICMP Information Request or Reply:	3.2.2.7				x		
ICMP Timestamp and Timestamp Reply:	3.2.2.8			x			
Minimize delay variability	3.2.2.8		x				1
Silently discard b'cast Timestamp	3.2.2.8			x			1
Silently discard m'cast Timestamp	3.2.2.8			x			1
Use specific-dest addr as TS Reply src	3.2.2.8	x					1
Reflect Record Route, Time Stamp options	3.2.2.6		x				1
Reverse and reflect Source Route option	3.2.2.8	x					1
Pass Timestamp Reply to higher layer	3.2.2.8	x					1
Obey rules for "standard value"	3.2.2.8	x					1
ICMP Address Mask Request and Reply:							
Addr Mask source configurable	3.2.2.9	x					
Support static configuration of addr mask	3.2.2.9	x					
Get addr mask dynamically during booting	3.2.2.9			x			
Get addr via ICMP Addr Mask Request/Reply	3.2.2.9			x			
Retransmit Addr Mask Req if no Reply	3.2.2.9	x					3
Assume default mask if no Reply	3.2.2.9		x				3
Update address mask from first Reply only	3.2.2.9	x					3
Reasonableness check on Addr Mask	3.2.2.9		x				
Send unauthorized Addr Mask Reply msgs	3.2.2.9					x	
Explicitly configured to be agent	3.2.2.9	x					
Static config=> Addr-Mask-Authoritative flag	3.2.2.9		x				
Broadcast Addr Mask Reply when init.	3.2.2.9	x					3
ROUTING OUTBOUND DATAGRAMS:							
Use address mask in local/remote decision	3.3.1.1	x					
Operate with no gateways on conn network	3.3.1.1	x					
Maintain "route cache" of next-hop gateways	3.3.1.2	x					
Treat Host and Net Redirect the same	3.3.1.2		x				
If no cache entry, use default gateway	3.3.1.2	x					
Support multiple default gateways	3.3.1.2	x					
Provide table of static routes	3.3.1.2			x			
Flag: route overridable by Redirects	3.3.1.2			x			
Key route cache on host, not net address	3.3.1.3			x			
Include TOS in route cache	3.3.1.3		x				
Able to detect failure of next-hop gateway	3.3.1.4	x					
Assume route is good forever	3.3.1.4				x		

Ping gateways continuously	3.3.1.4					x	
Ping only when traffic being sent	3.3.1.4	x					
Ping only when no positive indication	3.3.1.4	x					
Higher and lower layers give advice	3.3.1.4		x				
Switch from failed default g'way to another	3.3.1.5	x					
Manual method of entering config info	3.3.1.6	x					
REASSEMBLY and FRAGMENTATION:							
Able to reassemble incoming datagrams	3.3.2	x					
At least 576 byte datagrams	3.3.2	x					
EMTU_R configurable or indefinite	3.3.2		x				
Transport layer able to learn MMS_R	3.3.2	x					
Send ICMP Time Exceeded on reassembly timeout	3.3.2	x					
Fixed reassembly timeout value	3.3.2		x				
Pass MMS_S to higher layers	3.3.3	x					
Local fragmentation of outgoing packets	3.3.3			x			
Else don't send bigger than MMS_S	3.3.3	x					
Send max 576 to off-net destination	3.3.3		x				
All-Subnets-MTU configuration flag	3.3.3			x			
MULTIHOMING:							
Reply with same addr as spec-dest addr	3.3.4.2		x				
Allow application to choose local IP addr	3.3.4.2	x					
Silently discard d'gram in "wrong" interface	3.3.4.2			x			
Only send d'gram through "right" interface	3.3.4.2			x			4
SOURCE-ROUTE FORWARDING:							
Forward datagram with Source Route option	3.3.5			x			1
Obey corresponding gateway rules	3.3.5	x					1
Update TTL by gateway rules	3.3.5	x					1
Able to generate ICMP err code 4, 5	3.3.5	x					1
IP src addr not local host	3.3.5			x			1
Update Timestamp, Record Route options	3.3.5	x					1
Configurable switch for non-local SRing	3.3.5	x					1
Defaults to OFF	3.3.5	x					1
Satisfy gwy access rules for non-local SRing	3.3.5	x					1
If not forward, send Dest Unreach (cd 5)	3.3.5		x				2
BROADCAST:							
Broadcast addr as IP source addr	3.2.1.3					x	
Receive 0 or -1 broadcast formats OK	3.3.6		x				
Config'ble option to send 0 or -1 b'cast	3.3.6			x			
Default to -1 broadcast	3.3.6		x				
Recognize all broadcast address formats	3.3.6	x					
Use IP b'cast/m'cast addr in link-layer b'cast	3.3.6	x					
Silently discard link-layer-only b'cast dg's	3.3.6		x				
Use Limited Broadcast addr for connected net	3.3.6		x				

MULTICAST:						
Support local IP multicasting (RFC-1112)	3.3.7		x			
Support IGMP (RFC-1112)	3.3.7			x		
Join all-hosts group at startup	3.3.7		x			
Higher layers learn i'face m'cast capability	3.3.7		x			
INTERFACE:						
Allow transport layer to use all IP mechanisms	3.4	x				
Pass interface ident up to transport layer	3.4	x				
Pass all IP options up to transport layer	3.4	x				
Transport layer can send certain ICMP messages	3.4	x				
Pass spec'd ICMP messages up to transp. layer	3.4	x				
Include IP hdr+8 octets or more from orig.	3.4	x				
Able to leap tall buildings at a single bound	3.5		x			

Footnotes:

(1) Only if feature is implemented.

(2) This requirement is overruled if datagram is an ICMP error message.

(3) Only if feature is implemented and is configured "on".

(4) Unless has embedded gateway functionality or is source routed.

4. TRANSPORT PROTOCOLS

4.1 USER DATAGRAM PROTOCOL -- UDP

4.1.1 INTRODUCTION

The User Datagram Protocol UDP [UDP:1] offers only a minimal transport service -- non-guaranteed datagram delivery -- and gives applications direct access to the datagram service of the IP layer. UDP is used by applications that do not require the level of service of TCP or that wish to use communications services (e.g., multicast or broadcast delivery) not available from TCP.

UDP is almost a null protocol; the only services it provides over IP are checksumming of data and multiplexing by port number. Therefore, an application program running over UDP must deal directly with end-to-end communication problems that a connection-oriented protocol would have handled -- e.g., retransmission for reliable delivery, packetization and reassembly, flow control, congestion avoidance, etc., when these are required. The fairly complex coupling between IP and TCP will be mirrored in the coupling between UDP and many applications using UDP.

4.1.2 PROTOCOL WALK-THROUGH

There are no known errors in the specification of UDP.

4.1.3 SPECIFIC ISSUES

4.1.3.1 Ports

UDP well-known ports follow the same rules as TCP well-known ports; see Section 4.2.2.1 below.

If a datagram arrives addressed to a UDP port for which there is no pending LISTEN call, UDP SHOULD send an ICMP Port Unreachable message.

4.1.3.2 IP Options

UDP MUST pass any IP option that it receives from the IP layer transparently to the application layer.

An application MUST be able to specify IP options to be sent in its UDP datagrams, and UDP MUST pass these options to the IP layer.

DISCUSSION:

At present, the only options that need be passed through UDP are Source Route, Record Route, and Time Stamp. However, new options may be defined in the future, and UDP need not and should not make any assumptions about the format or content of options it passes to or from the application; an exception to this might be an IP-layer security option.

An application based on UDP will need to obtain a source route from a request datagram and supply a reversed route for sending the corresponding reply.

4.1.3.3 ICMP Messages

UDP MUST pass to the application layer all ICMP error messages that it receives from the IP layer. Conceptually at least, this may be accomplished with an upcall to the ERROR_REPORT routine (see Section 4.2.4.1).

DISCUSSION:

Note that ICMP error messages resulting from sending a UDP datagram are received asynchronously. A UDP-based application that wants to receive ICMP error messages is responsible for maintaining the state necessary to demultiplex these messages when they arrive; for example, the application may keep a pending receive operation for this purpose. The application is also responsible to avoid confusion from a delayed ICMP error message resulting from an earlier use of the same port(s).

4.1.3.4 UDP Checksums

A host MUST implement the facility to generate and validate UDP checksums. An application MAY optionally be able to control whether a UDP checksum will be generated, but it MUST default to checksumming on.

If a UDP datagram is received with a checksum that is non-zero and invalid, UDP MUST silently discard the datagram. An application MAY optionally be able to control whether UDP datagrams without checksums should be discarded or passed to the application.

DISCUSSION:

Some applications that normally run only across local area networks have chosen to turn off UDP checksums for efficiency. As a result, numerous cases of undetected errors have been reported. The advisability of ever turning off UDP checksumming is very controversial.

IMPLEMENTATION:

There is a common implementation error in UDP checksums. Unlike the TCP checksum, the UDP checksum is optional; the value zero is transmitted in the checksum field of a UDP header to indicate the absence of a checksum. If the transmitter really calculates a UDP checksum of zero, it must transmit the checksum as all 1's (65535). No special action is required at the receiver, since zero and 65535 are equivalent in 1's complement arithmetic.

4.1.3.5 UDP Multihoming

When a UDP datagram is received, its specific-destination address MUST be passed up to the application layer.

An application program MUST be able to specify the IP source address to be used for sending a UDP datagram or to leave it unspecified (in which case the networking software will choose an appropriate source address). There SHOULD be a way to communicate the chosen source address up to the application layer (e.g, so that the application can later receive a reply datagram only from the corresponding interface).

DISCUSSION:

A request/response application that uses UDP should use a source address for the response that is the same as the specific destination address of the request. See the "General Issues" section of [INTRO:1].

4.1.3.6 Invalid Addresses

A UDP datagram received with an invalid IP source address (e.g., a broadcast or multicast address) must be discarded by UDP or by the IP layer (see Section 3.2.1.3).

When a host sends a UDP datagram, the source address MUST be (one of) the IP address(es) of the host.

4.1.4 UDP/APPLICATION LAYER INTERFACE

The application interface to UDP MUST provide the full services of the IP/transport interface described in Section 3.4 of this document. Thus, an application using UDP needs the functions of the GET_SRCADDR(), GET_MAXSIZES(), ADVISE_DELIVPROB(), and RECV_ICMP() calls described in Section 3.4. For example, GET_MAXSIZES() can be used to learn the effective maximum UDP maximum datagram size for a particular {interface,remote host,TOS} triplet.

An application-layer program MUST be able to set the TTL and TOS values as well as IP options for sending a UDP datagram, and these values must be passed transparently to the IP layer. UDP MAY pass the received TOS up to the application layer.

4.1.5 UDP REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
UDP							
UDP send Port Unreachable	4.1.3.1		x				
IP Options in UDP							
- Pass rcv'd IP options to applic layer	4.1.3.2	x					
- Applic layer can specify IP options in Send	4.1.3.2	x					
- UDP passes IP options down to IP layer	4.1.3.2	x					
Pass ICMP msgs up to applic layer	4.1.3.3	x					
UDP checksums:							
- Able to generate/check checksum	4.1.3.4	x					
- Silently discard bad checksum	4.1.3.4	x					
- Sender Option to not generate checksum	4.1.3.4			x			
- Default is to checksum	4.1.3.4	x					
- Receiver Option to require checksum	4.1.3.4			x			
UDP Multihoming							
- Pass spec-dest addr to application	4.1.3.5	x					

- Applic layer can specify Local IP addr	4.1.3.5	x				
- Applic layer specify wild Local IP addr	4.1.3.5	x				
- Applic layer notified of Local IP addr used	4.1.3.5		x			
Bad IP src addr silently discarded by UDP/IP	4.1.3.6	x				
Only send valid IP source address	4.1.3.6	x				
UDP Application Interface Services						
Full IP interface of 3.4 for application	4.1.4	x				
- Able to spec TTL, TOS, IP opts when send dg	4.1.4	x				
- Pass received TOS up to applic layer	4.1.4			x		

4.2 TRANSMISSION CONTROL PROTOCOL -- TCP

4.2.1 INTRODUCTION

The Transmission Control Protocol TCP [TCP:1] is the primary virtual-circuit transport protocol for the Internet suite. TCP provides reliable, in-sequence delivery of a full-duplex stream of octets (8-bit bytes). TCP is used by those applications needing reliable, connection-oriented transport service, e.g., mail (SMTP), file transfer (FTP), and virtual terminal service (Telnet); requirements for these application-layer protocols are described in [INTRO:1].

4.2.2 PROTOCOL WALK-THROUGH

4.2.2.1 Well-Known Ports: RFC-793 Section 2.7

DISCUSSION:

TCP reserves port numbers in the range 0-255 for "well-known" ports, used to access services that are standardized across the Internet. The remainder of the port space can be freely allocated to application processes. Current well-known port definitions are listed in the RFC entitled "Assigned Numbers" [INTRO:6]. A prerequisite for defining a new well-known port is an RFC documenting the proposed service in enough detail to allow new implementations.

Some systems extend this notion by adding a third subdivision of the TCP port space: reserved ports, which are generally used for operating-system-specific services. For example, reserved ports might fall between 256 and some system-dependent upper limit. Some systems further choose to protect well-known and reserved ports by permitting only privileged users to open TCP connections with those port values. This is perfectly reasonable as long as the host does not assume that all hosts protect their low-numbered ports in this manner.

4.2.2.2 Use of Push: RFC-793 Section 2.8

When an application issues a series of SEND calls without setting the PUSH flag, the TCP MAY aggregate the data internally without sending it. Similarly, when a series of segments is received without the PSH bit, a TCP MAY queue the data internally without passing it to the receiving application.

The PSH bit is not a record marker and is independent of segment boundaries. The transmitter SHOULD collapse successive PSH bits when it packetizes data, to send the largest possible segment.

A TCP MAY implement PUSH flags on SEND calls. If PUSH flags are not implemented, then the sending TCP: (1) must not buffer data indefinitely, and (2) MUST set the PSH bit in the last buffered segment (i.e., when there is no more queued data to be sent).

The discussion in RFC-793 on pages 48, 50, and 74 erroneously implies that a received PSH flag must be passed to the application layer. Passing a received PSH flag to the application layer is now OPTIONAL.

An application program is logically required to set the PUSH flag in a SEND call whenever it needs to force delivery of the data to avoid a communication deadlock. However, a TCP SHOULD send a maximum-sized segment whenever possible, to improve performance (see Section 4.2.3.4).

DISCUSSION:

When the PUSH flag is not implemented on SEND calls, i.e., when the application/TCP interface uses a pure streaming model, responsibility for aggregating any tiny data fragments to form reasonable sized segments is partially borne by the application layer.

Generally, an interactive application protocol must set the PUSH flag at least in the last SEND call in each command or response sequence. A bulk transfer protocol like FTP should set the PUSH flag on the last segment of a file or when necessary to prevent buffer deadlock.

At the receiver, the PSH bit forces buffered data to be delivered to the application (even if less than a full buffer has been received). Conversely, the lack of a PSH bit can be used to avoid unnecessary wakeup calls to the application process; this can be an important performance optimization for large timesharing hosts. Passing the PSH bit to the receiving application allows an analogous optimization within the application.

4.2.2.3 Window Size: RFC-793 Section 3.1

The window size MUST be treated as an unsigned number, or else large window sizes will appear like negative windows and TCP will not work. It is RECOMMENDED that implementations reserve 32-bit fields for the send and receive window sizes in the connection record and do all window computations with 32 bits.

DISCUSSION:
It is known that the window field in the TCP header is too small for high-speed, long-delay paths. Experimental TCP options have been defined to extend the window size; see for example [TCP:11]. In anticipation of the adoption of such an extension, TCP implementors should treat windows as 32 bits.

4.2.2.4 Urgent Pointer: RFC-793 Section 3.1

The second sentence is in error: the urgent pointer points to the sequence number of the LAST octet (not LAST+1) in a sequence of urgent data. The description on page 56 (last sentence) is correct.

A TCP MUST support a sequence of urgent data of any length.

A TCP MUST inform the application layer asynchronously whenever it receives an Urgent pointer and there was previously no pending urgent data, or whenever the Urgent pointer advances in the data stream. There MUST be a way for the application to learn how much urgent data remains to be read from the connection, or at least to determine whether or not more urgent data remains to be read.

DISCUSSION:
Although the Urgent mechanism may be used for any application, it is normally used to send "interrupt"-type commands to a Telnet program (see "Using Telnet Synch Sequence" section in [INTRO:1]).

The asynchronous or "out-of-band" notification will allow the application to go into "urgent mode", reading data from the TCP connection. This allows control commands to be sent to an application whose normal input buffers are full of unprocessed data.

IMPLEMENTATION:
The generic ERROR-REPORT() upcall described in Section 4.2.4.1 is a possible mechanism for informing the application of the arrival of urgent data.

4.2.2.5 TCP Options: RFC-793 Section 3.1

A TCP MUST be able to receive a TCP option in any segment. A TCP MUST ignore without error any TCP option it does not implement, assuming that the option has a length field (all TCP options defined in the future will have length fields). TCP MUST be prepared to handle an illegal option length (e.g., zero) without crashing; a suggested procedure is to reset the connection and log the reason.

4.2.2.6 Maximum Segment Size Option: RFC-793 Section 3.1

TCP MUST implement both sending and receiving the Maximum Segment Size option [TCP:4].

TCP SHOULD send an MSS (Maximum Segment Size) option in every SYN segment when its receive MSS differs from the default 536, and MAY send it always.

If an MSS option is not received at connection setup, TCP MUST assume a default send MSS of 536 (576-40) [TCP:4].

The maximum size of a segment that TCP really sends, the "effective send MSS," MUST be the smaller of the send MSS (which reflects the available reassembly buffer size at the remote host) and the largest size permitted by the IP layer:

```
    Eff.snd.MSS =

        min(SendMSS+20, MMS_S) - TCPhdrsize - IPoptionsize
```

where:

* SendMSS is the MSS value received from the remote host, or the default 536 if no MSS option is received.

* MMS_S is the maximum size for a transport-layer message that TCP may send.

* TCPhdrsize is the size of the TCP header; this is normally 20, but may be larger if TCP options are to be sent.

* IPoptionsize is the size of any IP options that TCP will pass to the IP layer with the current message.

The MSS value to be sent in an MSS option must be less than or equal to:

MMS_R - 20

where MMS_R is the maximum size for a transport-layer message that can be received (and reassembled). TCP obtains MMS_R and MMS_S from the IP layer; see the generic call GET_MAXSIZES in Section 3.4.

DISCUSSION:

The choice of TCP segment size has a strong effect on performance. Larger segments increase throughput by amortizing header size and per-datagram processing overhead over more data bytes; however, if the packet is so large that it causes IP fragmentation, efficiency drops sharply if any fragments are lost [IP:9].

Some TCP implementations send an MSS option only if the destination host is on a non-connected network. However, in general the TCP layer may not have the appropriate information to make this decision, so it is preferable to leave to the IP layer the task of determining a suitable MTU for the Internet path. We therefore recommend that TCP always send the option (if not 536) and that the IP layer determine MMS_R as specified in 3.3.3 and 3.4. A proposed IP-layer mechanism to measure the MTU would then modify the IP layer without changing TCP.

4.2.2.7 TCP Checksum: RFC-793 Section 3.1

Unlike the UDP checksum (see Section 4.1.3.4), the TCP checksum is never optional. The sender MUST generate it and the receiver MUST check it.

4.2.2.8 TCP Connection State Diagram: RFC-793 Section 3.2, page 23

There are several problems with this diagram:

(a) The arrow from SYN-SENT to SYN-RCVD should be labeled with "snd SYN,ACK", to agree with the text on page 68 and with Figure 8.

(b) There could be an arrow from SYN-RCVD state to LISTEN state, conditioned on receiving a RST after a passive open (see text page 70).

(c) It is possible to go directly from FIN-WAIT-1 to the TIME-WAIT state (see page 75 of the spec).

4.2.2.9 Initial Sequence Number Selection: RFC-793 Section 3.3, page 27

A TCP MUST use the specified clock-driven selection of initial sequence numbers.

4.2.2.10 Simultaneous Open Attempts: RFC-793 Section 3.4, page 32

There is an error in Figure 8: the packet on line 7 should be identical to the packet on line 5.

A TCP MUST support simultaneous open attempts.

DISCUSSION:
It sometimes surprises implementors that if two applications attempt to simultaneously connect to each other, only one connection is generated instead of two. This was an intentional design decision; don't try to "fix" it.

4.2.2.11 Recovery from Old Duplicate SYN: RFC-793 Section 3.4, page 33

Note that a TCP implementation MUST keep track of whether a connection has reached SYN_RCVD state as the result of a passive OPEN or an active OPEN.

4.2.2.12 RST Segment: RFC-793 Section 3.4

A TCP SHOULD allow a received RST segment to include data.

DISCUSSION
It has been suggested that a RST segment could contain ASCII text that encoded and explained the cause of the RST. No standard has yet been established for such data.

4.2.2.13 Closing a Connection: RFC-793 Section 3.5

A TCP connection may terminate in two ways: (1) the normal TCP close sequence using a FIN handshake, and (2) an "abort" in which one or more RST segments are sent and the connection state is immediately discarded. If a TCP connection is closed by the remote site, the local application MUST be informed whether it closed normally or was aborted.

The normal TCP close sequence delivers buffered data reliably in both directions. Since the two directions of a TCP connection are closed independently, it is possible for a connection to be "half closed," i.e., closed in only one direction, and a host is permitted to continue sending data in the open direction on a half-closed connection.

A host MAY implement a "half-duplex" TCP close sequence, so that an application that has called CLOSE cannot continue to read data from the connection. If such a host issues a CLOSE call while received data is still pending in TCP, or if new data is received after CLOSE is called, its TCP SHOULD send a RST to show that data was lost.

When a connection is closed actively, it MUST linger in TIME-WAIT state for a time 2xMSL (Maximum Segment Lifetime). However, it MAY accept a new SYN from the remote TCP to reopen the connection directly from TIME-WAIT state, if it:

(1) assigns its initial sequence number for the new connection to be larger than the largest sequence number it used on the previous connection incarnation, and

(2) returns to TIME-WAIT state if the SYN turns out to be an old duplicate.

DISCUSSION:

TCP's full-duplex data-preserving close is a feature that is not included in the analogous ISO transport protocol TP4.

Some systems have not implemented half-closed connections, presumably because they do not fit into the I/O model of their particular operating system. On these systems, once an application has called CLOSE, it can no longer read input data from the connection; this is referred to as a "half-duplex" TCP close sequence.

The graceful close algorithm of TCP requires that the connection state remain defined on (at least) one end of the connection, for a timeout period of 2xMSL, i.e., 4 minutes. During this period, the (remote socket, local socket) pair that defines the connection is busy and cannot be reused. To shorten the time that a given port pair is tied up, some TCPs allow a new SYN to be accepted in TIME-WAIT state.

4.2.2.14 Data Communication: RFC-793 Section 3.7, page 40

Since RFC-793 was written, there has been extensive work on TCP algorithms to achieve efficient data communication. Later sections of the present document describe required and recommended TCP algorithms to determine when to send data (Section 4.2.3.4), when to send an acknowledgment (Section 4.2.3.2), and when to update the window (Section 4.2.3.3).

DISCUSSION:

One important performance issue is "Silly Window Syndrome" or "SWS" [TCP:5], a stable pattern of small incremental window movements resulting in extremely poor TCP performance. Algorithms to avoid SWS are described below for both the sending side (Section 4.2.3.4) and the receiving side (Section 4.2.3.3).

In brief, SWS is caused by the receiver advancing the right window edge whenever it has any new buffer space available to receive data and by the sender using any incremental window, no matter how small, to send more data [TCP:5]. The result can be a stable pattern of sending tiny data segments, even though both sender and receiver have a large total buffer space for the connection. SWS can only occur during the transmission of a large amount of data; if the connection goes quiescent, the problem will disappear. It is caused by typical straightforward implementation of window management, but the sender and receiver algorithms given below will avoid it.

Another important TCP performance issue is that some applications, especially remote login to character-at-a-time hosts, tend to send streams of one-octet data segments. To avoid deadlocks, every TCP SEND call from such applications must be "pushed", either explicitly by the application or else implicitly by TCP. The result may be a stream of TCP segments that contain one data octet each, which makes very inefficient use of the Internet and contributes to Internet congestion. The Nagle Algorithm described in Section 4.2.3.4 provides a simple and effective solution to this problem. It does have the effect of clumping characters over Telnet connections; this may initially surprise users accustomed to single-character echo, but user acceptance has not been a problem.

Note that the Nagle algorithm and the send SWS avoidance algorithm play complementary roles in improving performance. The Nagle algorithm discourages sending tiny segments when the data to be sent increases in small increments, while the SWS avoidance algorithm discourages small segments resulting from the right window edge advancing in small increments.

A careless implementation can send two or more acknowledgment segments per data segment received. For example, suppose the receiver acknowledges every data segment immediately. When the application program subsequently consumes the data and increases the available receive buffer space again, the receiver may send a second acknowledgment segment to update the window at the sender. The extreme case occurs with single-character segments on TCP connections using the Telnet protocol for remote login service. Some implementations have been observed in which each incoming 1-character segment generates three return segments: (1) the acknowledgment, (2) a one byte increase in the window, and (3) the echoed character, respectively.

4.2.2.15 Retransmission Timeout: RFC-793 Section 3.7, page 41

The algorithm suggested in RFC-793 for calculating the retransmission timeout is now known to be inadequate; see Section 4.2.3.1 below.

Recent work by Jacobson [TCP:7] on Internet congestion and TCP retransmission stability has produced a transmission algorithm combining "slow start" with "congestion avoidance". A TCP MUST implement this algorithm.

If a retransmitted packet is identical to the original packet (which implies not only that the data boundaries have not changed, but also that the window and acknowledgment fields of the header have not changed), then the same IP Identification field MAY be used (see Section 3.2.1.5).

IMPLEMENTATION:

Some TCP implementors have chosen to "packetize" the data stream, i.e., to pick segment boundaries when segments are originally sent and to queue these segments in a "retransmission queue" until they are acknowledged. Another design (which may be simpler) is to defer packetizing until each time data is transmitted or retransmitted, so there will be no segment retransmission queue.

In an implementation with a segment retransmission queue, TCP performance may be enhanced by repacketizing the segments awaiting acknowledgment when the first retransmission timeout occurs. That is, the outstanding segments that fitted would be combined into one maximum-sized segment, with a new IP Identification value. The TCP would then retain this combined segment in the retransmit queue until it was acknowledged. However, if the first two segments in the retransmission queue totalled more than one maximum-sized segment, the TCP would retransmit only the first segment using the original IP Identification field.

4.2.2.16 Managing the Window: RFC-793 Section 3.7, page 41

A TCP receiver SHOULD NOT shrink the window, i.e., move the right window edge to the left. However, a sending TCP MUST be robust against window shrinking, which may cause the "useable window" (see Section 4.2.3.4) to become negative.

If this happens, the sender SHOULD NOT send new data, but SHOULD retransmit normally the old unacknowledged data between SND.UNA and SND.UNA+SND.WND. The sender MAY also retransmit old data beyond SND.UNA+SND.WND, but SHOULD NOT time out the connection if data beyond the right window edge is not acknowledged. If the window shrinks to zero, the TCP MUST probe it in the standard way (see next Section).

DISCUSSION:

Many TCP implementations become confused if the window shrinks from the right after data has been sent into a larger window. Note that TCP has a heuristic to select the latest window update despite possible datagram reordering; as a result, it may ignore a window update with a smaller window than previously offered if neither the sequence number nor the acknowledgment number is increased.

4.2.2.17 Probing Zero Windows: RFC-793 Section 3.7, page 42

Probing of zero (offered) windows MUST be supported.

A TCP MAY keep its offered receive window closed indefinitely. As long as the receiving TCP continues to send acknowledgments in response to the probe segments, the sending TCP MUST allow the connection to stay open.

DISCUSSION:

It is extremely important to remember that ACK (acknowledgment) segments that contain no data are not reliably transmitted by TCP. If zero window probing is not supported, a connection may hang forever when an ACK segment that re-opens the window is lost.

The delay in opening a zero window generally occurs when the receiving application stops taking data from its TCP. For example, consider a printer daemon application, stopped because the printer ran out of paper.

The transmitting host SHOULD send the first zero-window probe when a zero window has existed for the retransmission timeout period (see Section 4.2.2.15), and SHOULD increase exponentially the interval between successive probes.

DISCUSSION:

This procedure minimizes delay if the zero-window condition is due to a lost ACK segment containing a window-opening update. Exponential backoff is recommended, possibly with some maximum interval not specified here. This procedure is similar to that of the retransmission algorithm, and it may be possible to combine the two procedures in the implementation.

4.2.2.18 Passive OPEN Calls: RFC-793 Section 3.8

Every passive OPEN call either creates a new connection record in LISTEN state, or it returns an error; it MUST NOT affect any previously created connection record.

A TCP that supports multiple concurrent users MUST provide an OPEN call that will functionally allow an application to LISTEN on a port while a connection block with the same local port is in SYN-SENT or SYN-RECEIVED state.

DISCUSSION:

Some applications (e.g., SMTP servers) may need to handle multiple connection attempts at about the same time. The probability of a connection attempt failing is reduced by giving the application some means of listening for a new connection at the same time that an earlier connection attempt is going through the three-way handshake.

IMPLEMENTATION:
Acceptable implementations of concurrent opens may permit multiple passive OPEN calls, or they may allow "cloning" of LISTEN-state connections from a single passive OPEN call.

4.2.2.19 Time to Live: RFC-793 Section 3.9, page 52

RFC-793 specified that TCP was to request the IP layer to send TCP segments with TTL = 60. This is obsolete; the TTL value used to send TCP segments MUST be configurable. See Section 3.2.1.7 for discussion.

4.2.2.20 Event Processing: RFC-793 Section 3.9

While it is not strictly required, a TCP SHOULD be capable of queueing out-of-order TCP segments. Change the "may" in the last sentence of the first paragraph on page 70 to "should".

DISCUSSION:
Some small-host implementations have omitted segment queueing because of limited buffer space. This omission may be expected to adversely affect TCP throughput, since loss of a single segment causes all later segments to appear to be "out of sequence".

In general, the processing of received segments MUST be implemented to aggregate ACK segments whenever possible. For example, if the TCP is processing a series of queued segments, it MUST process them all before sending any ACK segments.

Here are some detailed error corrections and notes on the Event Processing section of RFC-793.

(a) CLOSE Call, CLOSE-WAIT state, p. 61: enter LAST-ACK state, not CLOSING.

(b) LISTEN state, check for SYN (pp. 65, 66): With a SYN bit, if the security/compartment or the precedence is wrong for the segment, a reset is sent. The wrong form of reset is shown in the text; it should be:

```
<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>
```

(c) SYN-SENT state, Check for SYN, p. 68: When the connection enters ESTABLISHED state, the following variables must be set:

```
SND.WND <- SEG.WND
SND.WL1 <- SEG.SEQ
SND.WL2 <- SEG.ACK
```

(d) Check security and precedence, p. 71: The first heading "ESTABLISHED STATE" should really be a list of all states other than SYN-RECEIVED: ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, and TIME-WAIT.

(e) Check SYN bit, p. 71: "In SYN-RECEIVED state and if the connection was initiated with a passive OPEN, then return this connection to the LISTEN state and return. Otherwise...".

(f) Check ACK field, SYN-RECEIVED state, p. 72: When the connection enters ESTABLISHED state, the variables listed in (c) must be set.

(g) Check ACK field, ESTABLISHED state, p. 72: The ACK is a duplicate if SEG.ACK =< SND.UNA (the = was omitted). Similarly, the window should be updated if: SND.UNA =< SEG.ACK =< SND.NXT.

(h) USER TIMEOUT, p. 77:

It would be better to notify the application of the timeout rather than letting TCP force the connection closed. However, see also Section 4.2.3.5.

4.2.2.21 Acknowledging Queued Segments: RFC-793 Section 3.9

A TCP MAY send an ACK segment acknowledging RCV.NXT when a valid segment arrives that is in the window but not at the left window edge.

DISCUSSION:

RFC-793 (see page 74) was ambiguous about whether or not an ACK segment should be sent when an out-of-order segment was received, i.e., when SEG.SEQ was unequal to RCV.NXT.

One reason for ACKing out-of-order segments might be to support an experimental algorithm known as "fast retransmit". With this algorithm, the sender uses the "redundant" ACK's to deduce that a segment has been lost before the retransmission timer has expired. It counts the number of times an ACK has been received with the same value of SEG.ACK and with the same right window edge. If more than a threshold number of such ACK's is received, then the segment containing the octets starting at SEG.ACK is assumed to have been lost and is retransmitted, without awaiting a timeout. The threshold is chosen to compensate for the maximum likely segment reordering in the Internet. There is not yet enough experience with the fast retransmit algorithm to determine how useful it is.

4.2.3 SPECIFIC ISSUES

4.2.3.1 Retransmission Timeout Calculation

A host TCP MUST implement Karn's algorithm and Jacobson's algorithm for computing the retransmission timeout ("RTO").

- Jacobson's algorithm for computing the smoothed round-trip ("RTT") time incorporates a simple measure of the variance [TCP:7].
- Karn's algorithm for selecting RTT measurements ensures that ambiguous round-trip times will not corrupt the calculation of the smoothed round-trip time [TCP:6].

This implementation also MUST include "exponential backoff" for successive RTO values for the same segment. Retransmission of SYN segments SHOULD use the same algorithm as data segments.

DISCUSSION:

There were two known problems with the RTO calculations specified in RFC-793. First, the accurate measurement of RTTs is difficult when there are retransmissions. Second, the algorithm to compute the smoothed round-trip time is inadequate [TCP:7], because it incorrectly assumed that the variance in RTT values would be small and constant. These problems were solved by Karn's and Jacobson's algorithm, respectively.

The performance increase resulting from the use of these improvements varies from noticeable to dramatic. Jacobson's algorithm for incorporating the measured RTT variance is especially important on a low-speed link, where the natural variation of packet sizes causes a large variation in RTT. One vendor found link utilization on a 9.6kb line went from 10% to 90% as a result of implementing Jacobson's variance algorithm in TCP.

The following values SHOULD be used to initialize the estimation parameters for a new connection:

(a) RTT = 0 seconds.

(b) RTO = 3 seconds. (The smoothed variance is to be initialized to the value that will result in this RTO).

The recommended upper and lower bounds on the RTO are known to be inadequate on large internets. The lower bound SHOULD be measured in fractions of a second (to accommodate high speed LANs) and the upper bound should be 2*MSL, i.e., 240 seconds.

DISCUSSION:

Experience has shown that these initialization values are reasonable, and that in any case the Karn and Jacobson algorithms make TCP behavior reasonably insensitive to the initial parameter choices.

4.2.3.2 When to Send an ACK Segment

A host that is receiving a stream of TCP data segments can increase efficiency in both the Internet and the hosts by sending fewer than one ACK (acknowledgment) segment per data segment received; this is known as a "delayed ACK" [TCP:5].

A TCP SHOULD implement a delayed ACK, but an ACK should not be excessively delayed; in particular, the delay MUST be less than 0.5 seconds, and in a stream of full-sized segments there SHOULD be an ACK for at least every second segment.

DISCUSSION:

A delayed ACK gives the application an opportunity to update the window and perhaps to send an immediate response. In particular, in the case of character-mode remote login, a delayed ACK can reduce the number of segments sent by the server by a factor of 3 (ACK, window update, and echo character all combined in one segment).

In addition, on some large multi-user hosts, a delayed ACK can substantially reduce protocol processing overhead by reducing the total number of packets to be processed [TCP:5]. However, excessive delays on ACK's can disturb the round-trip timing and packet "clocking" algorithms [TCP:7].

4.2.3.3 When to Send a Window Update

A TCP MUST include a SWS avoidance algorithm in the receiver [TCP:5].

IMPLEMENTATION:

The receiver's SWS avoidance algorithm determines when the right window edge may be advanced; this is customarily known as "updating the window". This algorithm combines with the delayed ACK algorithm (see Section 4.2.3.2) to determine when an ACK segment containing the current window will really be sent to the receiver. We use the notation of RFC-793; see Figures 4 and 5 in that document.

The solution to receiver SWS is to avoid advancing the right window edge RCV.NXT+RCV.WND in small increments, even if data is received from the network in small segments.

Suppose the total receive buffer space is RCV.BUFF. At any given moment, RCV.USER octets of this total may be tied up with data that has been received and acknowledged but which the user process has not yet consumed. When the connection is quiescent, RCV.WND = RCV.BUFF and RCV.USER = 0.

Keeping the right window edge fixed as data arrives and is acknowledged requires that the receiver offer less than its full buffer space, i.e., the receiver must specify a RCV.WND that keeps RCV.NXT+RCV.WND constant as RCV.NXT increases. Thus, the total buffer space RCV.BUFF is generally divided into three parts:

```
    |<------- RCV.BUFF ---------------->|
         1             2            3
----|---------|------------------|------|----
          RCV.NXT                ^
                              (Fixed)
```

1 - RCV.USER = data received but not yet consumed;
2 - RCV.WND = space advertised to sender;
3 - Reduction = space available but not yet advertised.

The suggested SWS avoidance algorithm for the receiver is to keep RCV.NXT+RCV.WND fixed until the reduction satisfies:

$$\text{RCV.BUFF} - \text{RCV.USER} - \text{RCV.WND} \geq \min(\text{Fr} * \text{RCV.BUFF}, \text{Eff.snd.MSS})$$

where Fr is a fraction whose recommended value is 1/2, and Eff.snd.MSS is the effective send MSS for the connection (see Section 4.2.2.6). When the inequality is satisfied, RCV.WND is set to RCV.BUFF-RCV.USER.

Note that the general effect of this algorithm is to advance RCV.WND in increments of Eff.snd.MSS (for realistic receive buffers: Eff.snd.MSS < RCV.BUFF/2). Note also that the receiver must use its own Eff.snd.MSS, assuming it is the same as the sender's.

4.2.3.4 When to Send Data

A TCP MUST include a SWS avoidance algorithm in the sender.

A TCP SHOULD implement the Nagle Algorithm [TCP:9] to coalesce short segments. However, there MUST be a way for an application to disable the Nagle algorithm on an individual connection. In all cases, sending data is also subject to the limitation imposed by the Slow Start algorithm (Section 4.2.2.15).

DISCUSSION:
The Nagle algorithm is generally as follows:

If there is unacknowledged data (i.e., SND.NXT > SND.UNA), then the sending TCP buffers all user data (regardless of the PSH bit), until the outstanding data has been acknowledged or until the TCP can send a full-sized segment (Eff.snd.MSS bytes; see Section 4.2.2.6).

Some applications (e.g., real-time display window updates) require that the Nagle algorithm be turned off, so small data segments can be streamed out at the maximum rate.

IMPLEMENTATION:
The sender's SWS avoidance algorithm is more difficult than the receivers's, because the sender does not know (directly) the receiver's total buffer space RCV.BUFF. An approach which has been found to work well is for the sender to calculate Max(SND.WND), the maximum send window it has seen so far on the connection, and to use this value as an estimate of RCV.BUFF. Unfortunately, this can only be an estimate; the receiver may at any time reduce the size of RCV.BUFF. To avoid a resulting deadlock, it is necessary to have a timeout to force transmission of data, overriding the SWS avoidance algorithm. In practice, this timeout should seldom occur.

The "useable window" [TCP:5] is:

U = SND.UNA + SND.WND - SND.NXT

i.e., the offered window less the amount of data sent but not acknowledged. If D is the amount of data queued in the sending TCP but not yet sent, then the following set of rules is recommended.

Send data:

(1) if a maximum-sized segment can be sent, i.e, if:

min(D,U) >= Eff.snd.MSS;

(2) or if the data is pushed and all queued data can be sent now, i.e., if:

[SND.NXT = SND.UNA and] PUSHED and D <= U

(the bracketed condition is imposed by the Nagle algorithm);

(3) or if at least a fraction Fs of the maximum window can be sent, i.e., if:

```
        [SND.NXT = SND.UNA and]

                  min(D.U) >= Fs * Max(SND.WND);
```

(4) or if data is PUSHed and the override timeout occurs.

Here Fs is a fraction whose recommended value is 1/2. The override timeout should be in the range 0.1 - 1.0 seconds. It may be convenient to combine this timer with the timer used to probe zero windows (Section 4.2.2.17).

Finally, note that the SWS avoidance algorithm just specified is to be used instead of the sender-side algorithm contained in [TCP:5].

4.2.3.5 TCP Connection Failures

Excessive retransmission of the same segment by TCP indicates some failure of the remote host or the Internet path. This failure may be of short or long duration. The following procedure MUST be used to handle excessive retransmissions of data segments [IP:11]:

(a) There are two thresholds R1 and R2 measuring the amount of retransmission that has occurred for the same segment. R1 and R2 might be measured in time units or as a count of retransmissions.

(b) When the number of transmissions of the same segment reaches or exceeds threshold R1, pass negative advice (see Section 3.3.1.4) to the IP layer, to trigger dead-gateway diagnosis.

(c) When the number of transmissions of the same segment reaches a threshold R2 greater than R1, close the connection.

(d) An application MUST be able to set the value for R2 for a particular connection. For example, an interactive application might set R2 to "infinity," giving the user control over when to disconnect.

(d) TCP SHOULD inform the application of the delivery problem (unless such information has been disabled by the application; see Section 4.2.4.1), when R1 is reached and before R2. This will allow a remote login (User Telnet) application program to inform the user, for example.

The value of R1 SHOULD correspond to at least 3 retransmissions, at the current RTO. The value of R2 SHOULD correspond to at least 100 seconds.

An attempt to open a TCP connection could fail with excessive retransmissions of the SYN segment or by receipt of a RST segment or an ICMP Port Unreachable. SYN retransmissions MUST be handled in the general way just described for data retransmissions, including notification of the application layer.

However, the values of R1 and R2 may be different for SYN and data segments. In particular, R2 for a SYN segment MUST be set large enough to provide retransmission of the segment for at least 3 minutes. The application can close the connection (i.e., give up on the open attempt) sooner, of course.

DISCUSSION:

Some Internet paths have significant setup times, and the number of such paths is likely to increase in the future.

4.2.3.6 TCP Keep-Alives

Implementors MAY include "keep-alives" in their TCP implementations, although this practice is not universally accepted. If keep-alives are included, the application MUST be able to turn them on or off for each TCP connection, and they MUST default to off.

Keep-alive packets MUST only be sent when no data or acknowledgement packets have been received for the connection within an interval. This interval MUST be configurable and MUST default to no less than two hours.

It is extremely important to remember that ACK segments that contain no data are not reliably transmitted by TCP. Consequently, if a keep-alive mechanism is implemented it MUST NOT interpret failure to respond to any specific probe as a dead connection.

An implementation SHOULD send a keep-alive segment with no data; however, it MAY be configurable to send a keep-alive segment containing one garbage octet, for compatibility with erroneous TCP implementations.

DISCUSSION:

A "keep-alive" mechanism periodically probes the other end of a connection when the connection is otherwise idle, even when there is no data to be sent. The TCP specification does not include a keep-alive mechanism because it could: (1) cause perfectly good connections to break during transient Internet failures; (2) consume unnecessary bandwidth ("if no one is using the connection, who cares if it is still good?"); and (3) cost money for an Internet path that charges for packets.

Some TCP implementations, however, have included a keep-alive mechanism. To confirm that an idle connection is still active, these implementations send a probe segment designed to elicit a response from the peer TCP. Such a segment generally contains SEG.SEQ = SND.NXT-1 and may or may not contain one garbage octet of data. Note that on a quiet connection SND.NXT = RCV.NXT, so that this SEG.SEQ will be outside the window. Therefore, the probe causes the receiver to return an acknowledgment segment, confirming that the connection is still live. If the peer has dropped the connection due to a network partition or a crash, it will respond with a RST instead of an acknowledgment segment.

Unfortunately, some misbehaved TCP implementations fail to respond to a segment with SEG.SEQ = SND.NXT-1 unless the segment contains data. Alternatively, an implementation could determine whether a peer responded correctly to keep-alive packets with no garbage data octet.

A TCP keep-alive mechanism should only be invoked in server applications that might otherwise hang indefinitely and consume resources unnecessarily if a client crashes or aborts a connection during a network failure.

4.2.3.7 TCP Multihoming

If an application on a multihomed host does not specify the local IP address when actively opening a TCP connection, then the TCP MUST ask the IP layer to select a local IP address before sending the (first) SYN. See the function GET_SRCADDR() in Section 3.4.

At all other times, a previous segment has either been sent or received on this connection, and TCP MUST use the same local address is used that was used in those previous segments.

4.2.3.8 IP Options

When received options are passed up to TCP from the IP layer, TCP MUST ignore options that it does not understand.

A TCP MAY support the Time Stamp and Record Route options.

An application MUST be able to specify a source route when it actively opens a TCP connection, and this MUST take precedence over a source route received in a datagram.

When a TCP connection is OPENed passively and a packet arrives with a completed IP Source Route option (containing a return route), TCP MUST save the return route and use it for all segments sent on this connection. If a different source route arrives in a later segment, the later definition SHOULD override the earlier one.

4.2.3.9 ICMP Messages

TCP MUST act on an ICMP error message passed up from the IP layer, directing it to the connection that created the error. The necessary demultiplexing information can be found in the IP header contained within the ICMP message.

- Source Quench

  TCP MUST react to a Source Quench by slowing transmission on the connection. The RECOMMENDED procedure is for a Source Quench to trigger a "slow start," as if a retransmission timeout had occurred.

- Destination Unreachable -- codes 0, 1, 5

  Since these Unreachable messages indicate soft error conditions, TCP MUST NOT abort the connection, and it SHOULD make the information available to the application.

DISCUSSION:

TCP could report the soft error condition directly to the application layer with an upcall to the ERROR_REPORT routine, or it could merely note the message and report it to the application only when and if the TCP connection times out.

- Destination Unreachable -- codes 2-4

  These are hard error conditions, so TCP SHOULD abort the connection.

- Time Exceeded -- codes 0, 1

  This should be handled the same way as Destination Unreachable codes 0, 1, 5 (see above).

- Parameter Problem

  This should be handled the same way as Destination Unreachable codes 0, 1, 5 (see above).

4.2.3.10 Remote Address Validation

A TCP implementation MUST reject as an error a local OPEN call for an invalid remote IP address (e.g., a broadcast or multicast address).

An incoming SYN with an invalid source address must be ignored either by TCP or by the IP layer (see Section 3.2.1.3).

A TCP implementation MUST silently discard an incoming SYN segment that is addressed to a broadcast or multicast address.

4.2.3.11 TCP Traffic Patterns

IMPLEMENTATION:

The TCP protocol specification [TCP:1] gives the implementor much freedom in designing the algorithms that control the message flow over the connection -- packetizing, managing the window, sending acknowledgments, etc. These design decisions are difficult because a TCP must adapt to a wide range of traffic patterns. Experience has shown that a TCP implementor needs to verify the design on two extreme traffic patterns:

- Single-character Segments

  Even if the sender is using the Nagle Algorithm, when a TCP connection carries remote login traffic across a low-delay LAN the receiver will generally get a stream of single-character segments. If remote terminal echo mode is in effect, the receiver's system will generally echo each character as it is received.

- Bulk Transfer

  When TCP is used for bulk transfer, the data stream should be made up (almost) entirely of segments of the size of the effective MSS. Although TCP uses a sequence number space with byte (octet) granularity, in bulk-transfer mode its operation should be as if TCP used a sequence space that counted only segments.

Experience has furthermore shown that a single TCP can effectively and efficiently handle these two extremes.

The most important tool for verifying a new TCP implementation is a packet trace program. There is a large volume of experience showing the importance of tracing a variety of traffic patterns with other TCP implementations and studying the results carefully.

4.2.3.12 Efficiency

IMPLEMENTATION:

Extensive experience has led to the following suggestions for efficient implementation of TCP:

(a) Don't Copy Data

In bulk data transfer, the primary CPU-intensive tasks are copying data from one place to another and checksumming the data. It is vital to minimize the number of copies of TCP data. Since the ultimate speed limitation may be fetching data across the memory bus, it may be useful to combine the copy with checksumming, doing both with a single memory fetch.

(b) Hand-Craft the Checksum Routine

A good TCP checksumming routine is typically two to five times faster than a simple and direct implementation of the definition. Great care and clever coding are often required and advisable to make the checksumming code "blazing fast". See [TCP:10].

(c) Code for the Common Case

TCP protocol processing can be complicated, but for most segments there are only a few simple decisions to be made. Per-segment processing will be greatly speeded up by coding the main line to minimize the number of decisions in the most common case.

4.2.4 TCP/APPLICATION LAYER INTERFACE

4.2.4.1 Asynchronous Reports

There MUST be a mechanism for reporting soft TCP error conditions to the application. Generically, we assume this takes the form of an application-supplied ERROR_REPORT routine that may be upcalled [INTRO:7] asynchronously from the transport layer:

```
ERROR_REPORT(local connection name, reason, subreason)
```

The precise encoding of the reason and subreason parameters is not specified here. However, the conditions that are reported asynchronously to the application MUST include:

- ICMP error message arrived (see 4.2.3.9)
- Excessive retransmissions (see 4.2.3.5)
- Urgent pointer advance (see 4.2.2.4).

However, an application program that does not want to receive such ERROR_REPORT calls SHOULD be able to effectively disable these calls.

DISCUSSION:

These error reports generally reflect soft errors that can be ignored without harm by many applications. It has been suggested that these error report calls should default to "disabled," but this is not required.

4.2.4.2 Type-of-Service

The application layer MUST be able to specify the Type-of-Service (TOS) for segments that are sent on a connection. It not required, but the application SHOULD be able to change the TOS during the connection lifetime. TCP SHOULD pass the current TOS value without change to the IP layer, when it sends segments on the connection.

The TOS will be specified independently in each direction on the connection, so that the receiver application will specify the TOS used for ACK segments.

TCP MAY pass the most recently received TOS up to the application.

DISCUSSION

Some applications (e.g., SMTP) change the nature of their communication during the lifetime of a connection, and therefore would like to change the TOS specification.

Note also that the OPEN call specified in RFC-793 includes a parameter ("options") in which the caller can specify IP options such as source route, record route, or timestamp.

4.2.4.3 Flush Call

Some TCP implementations have included a FLUSH call, which will empty the TCP send queue of any data for which the user has issued SEND calls but which is still to the right of the current send window. That is, it flushes as much queued send data as possible without losing sequence number synchronization. This is useful for implementing the "abort output" function of Telnet.

4.2.4.4 Multihoming

The user interface outlined in sections 2.7 and 3.8 of RFC-793 needs to be extended for multihoming. The OPEN call MUST have an optional parameter:

```
OPEN( ... [local IP address,] ... )
```

to allow the specification of the local IP address.

DISCUSSION:

Some TCP-based applications need to specify the local IP address to be used to open a particular connection; FTP is an example.

IMPLEMENTATION:

A passive OPEN call with a specified "local IP address" parameter will await an incoming connection request to that address. If the parameter is unspecified, a passive OPEN will await an incoming connection request to any local IP address, and then bind the local IP address of the connection to the particular address that is used.

For an active OPEN call, a specified "local IP address" parameter will be used for opening the connection. If the parameter is unspecified, the networking software will choose an appropriate local IP address (see Section 3.3.4.2) for the connection

4.2.5 TCP REQUIREMENT SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Push flag							
Aggregate or queue un-pushed data	4.2.2.2			x			
Sender collapse successive PSH flags	4.2.2.2		x				
SEND call can specify PUSH	4.2.2.2			x			

If cannot: sender buffer indefinitely	4.2.2.2					x	
If cannot: PSH last segment	4.2.2.2	x					
Notify receiving ALP of PSH	4.2.2.2			x			1
Send max size segment when possible	4.2.2.2		x				
Window							
Treat as unsigned number	4.2.2.3	x					
Handle as 32-bit number	4.2.2.3		x				
Shrink window from right	4.2.2.16				x		
Robust against shrinking window	4.2.2.16	x					
Receiver's window closed indefinitely	4.2.2.17			x			
Sender probe zero window	4.2.2.17	x					
First probe after RTO	4.2.2.17		x				
Exponential backoff	4.2.2.17		x				
Allow window stay zero indefinitely	4.2.2.17	x					
Sender timeout OK conn with zero wind	4.2.2.17					x	
Urgent Data							
Pointer points to last octet	4.2.2.4	x					
Arbitrary length urgent data sequence	4.2.2.4	x					
Inform ALP asynchronously of urgent data	4.2.2.4	x					1
ALP can learn if/how much urgent data Q'd	4.2.2.4	x					1
TCP Options							
Receive TCP option in any segment	4.2.2.5	x					
Ignore unsupported options	4.2.2.5	x					
Cope with illegal option length	4.2.2.5	x					
Implement sending & receiving MSS option	4.2.2.6	x					
Send MSS option unless 536	4.2.2.6		x				
Send MSS option always	4.2.2.6			x			
Send-MSS default is 536	4.2.2.6	x					
Calculate effective send seg size	4.2.2.6	x					
TCP Checksums							
Sender compute checksum	4.2.2.7	x					
Receiver check checksum	4.2.2.7	x					
Use clock-driven ISN selection	4.2.2.9	x					
Opening Connections							
Support simultaneous open attempts	4.2.2.10	x					
SYN-RCVD remembers last state	4.2.2.11	x					
Passive Open call interfere with others	4.2.2.18					x	
Function: simultan. LISTENs for same port	4.2.2.18	x					
Ask IP for src address for SYN if necc.	4.2.3.7	x					
Otherwise, use local addr of conn.	4.2.3.7	x					
OPEN to broadcast/multicast IP Address	4.2.3.14					x	
Silently discard seg to bcast/mcast addr	4.2.3.14	x					

Closing Connections							
RST can contain data	4.2.2.12		x				
Inform application of aborted conn	4.2.2.13	x					
Half-duplex close connections	4.2.2.13			x			
Send RST to indicate data lost	4.2.2.13		x				
In TIME-WAIT state for 2xMSL seconds	4.2.2.13	x					
Accept SYN from TIME-WAIT state	4.2.2.13			x			
Retransmissions							
Jacobson Slow Start algorithm	4.2.2.15	x					
Jacobson Congestion-Avoidance algorithm	4.2.2.15	x					
Retransmit with same IP ident	4.2.2.15			x			
Karn's algorithm	4.2.3.1	x					
Jacobson's RTO estimation alg.	4.2.3.1	x					
Exponential backoff	4.2.3.1	x					
SYN RTO calc same as data	4.2.3.1		x				
Recommended initial values and bounds	4.2.3.1		x				
Generating ACK's:							
Queue out-of-order segments	4.2.2.20		x				
Process all Q'd before send ACK	4.2.2.20	x					
Send ACK for out-of-order segment	4.2.2.21			x			
Delayed ACK's	4.2.3.2		x				
Delay < 0.5 seconds	4.2.3.2	x					
Every 2nd full-sized segment ACK'd	4.2.3.2	x					
Receiver SWS-Avoidance Algorithm	4.2.3.3	x					
Sending data							
Configurable TTL	4.2.2.19	x					
Sender SWS-Avoidance Algorithm	4.2.3.4	x					
Nagle algorithm	4.2.3.4		x				
Application can disable Nagle algorithm	4.2.3.4	x					
Connection Failures:							
Negative advice to IP on R1 retxs	4.2.3.5	x					
Close connection on R2 retxs	4.2.3.5	x					
ALP can set R2	4.2.3.5	x					1
Inform ALP of R1<=retxs<R2	4.2.3.5		x				1
Recommended values for R1, R2	4.2.3.5		x				
Same mechanism for SYNs	4.2.3.5	x					
R2 at least 3 minutes for SYN	4.2.3.5	x					
Send Keep-alive Packets:	4.2.3.6			x			
- Application can request	4.2.3.6	x					
- Default is "off"	4.2.3.6	x					
- Only send if idle for interval	4.2.3.6	x					
- Interval configurable	4.2.3.6	x					

- Default at least 2 hrs.	4.2.3.6	x					
- Tolerant of lost ACK's	4.2.3.6	x					
IP Options							
Ignore options TCP doesn't understand	4.2.3.8	x					
Time Stamp support	4.2.3.8			x			
Record Route support	4.2.3.8			x			
Source Route:							
ALP can specify	4.2.3.8	x					1
Overrides src rt in datagram	4.2.3.8	x					
Build return route from src rt	4.2.3.8	x					
Later src route overrides	4.2.3.8		x				
Receiving ICMP Messages from IP	4.2.3.9	x					
Dest. Unreach (0,1,5) => inform ALP	4.2.3.9		x				
Dest. Unreach (0,1,5) => abort conn	4.2.3.9					x	
Dest. Unreach (2-4) => abort conn	4.2.3.9		x				
Source Quench => slow start	4.2.3.9		x				
Time Exceeded => tell ALP, don't abort	4.2.3.9		x				
Param Problem => tell ALP, don't abort	4.2.3.9		x				
Address Validation							
Reject OPEN call to invalid IP address	4.2.3.10	x					
Reject SYN from invalid IP address	4.2.3.10	x					
Silently discard SYN to bcast/mcast addr	4.2.3.10	x					
TCP/ALP Interface Services							
Error Report mechanism	4.2.4.1	x					
ALP can disable Error Report Routine	4.2.4.1		x				
ALP can specify TOS for sending	4.2.4.2	x					
Passed unchanged to IP	4.2.4.2		x				
ALP can change TOS during connection	4.2.4.2		x				
Pass received TOS up to ALP	4.2.4.2			x			
FLUSH call	4.2.4.3			x			
Optional local IP addr parm. in OPEN	4.2.4.4	x					

FOOTNOTES:

(1) "ALP" means Application-Layer program.

5. REFERENCES

INTRODUCTORY REFERENCES

[INTRO:1] "Requirements for Internet Hosts -- Application and Support," IETF Host Requirements Working Group, R. Braden, Ed., RFC-1123, October 1989.

[INTRO:2] "Requirements for Internet Gateways," R. Braden and J. Postel, RFC-1009, June 1987.

[INTRO:3] "DDN Protocol Handbook," NIC-50004, NIC-50005, NIC-50006, (three volumes), SRI International, December 1985.

[INTRO:4] "Official Internet Protocols," J. Reynolds and J. Postel, RFC-1011, May 1987.

This document is republished periodically with new RFC numbers; the latest version must be used.

[INTRO:5] "Protocol Document Order Information," O. Jacobsen and J. Postel, RFC-980, March 1986.

[INTRO:6] "Assigned Numbers," J. Reynolds and J. Postel, RFC-1010, May 1987.

This document is republished periodically with new RFC numbers; the latest version must be used.

[INTRO:7] "Modularity and Efficiency in Protocol Implementations," D. Clark, RFC-817, July 1982.

[INTRO:8] "The Structuring of Systems Using Upcalls," D. Clark, 10th ACM SOSP, Orcas Island, Washington, December 1985.

Secondary References:

[INTRO:9] "A Protocol for Packet Network Intercommunication," V. Cerf and R. Kahn, IEEE Transactions on Communication, May 1974.

[INTRO:10] "The ARPA Internet Protocol," J. Postel, C. Sunshine, and D. Cohen, Computer Networks, Vol. 5, No. 4, July 1981.

[INTRO:11] "The DARPA Internet Protocol Suite," B. Leiner, J. Postel, R. Cole and D. Mills, Proceedings INFOCOM 85, IEEE, Washington DC, March 1985. Also in: IEEE Communications Magazine, March 1985. Also available as ISI-RS-85-153.

[INTRO:12] "Final Text of DIS8473, Protocol for Providing the Connectionless Mode Network Service," ANSI, published as RFC-994, March 1986.

[INTRO:13] "End System to Intermediate System Routing Exchange Protocol," ANSI X3S3.3, published as RFC-995, April 1986.

LINK LAYER REFERENCES

[LINK:1] "Trailer Encapsulations," S. Leffler and M. Karels, RFC-893, April 1984.

[LINK:2] "An Ethernet Address Resolution Protocol," D. Plummer, RFC-826, November 1982.

[LINK:3] "A Standard for the Transmission of IP Datagrams over Ethernet Networks," C. Hornig, RFC-894, April 1984.

[LINK:4] "A Standard for the Transmission of IP Datagrams over IEEE 802 "Networks," J. Postel and J. Reynolds, RFC-1042, February 1988.

This RFC contains a great deal of information of importance to Internet implementers planning to use IEEE 802 networks.

IP LAYER REFERENCES

[IP:1] "Internet Protocol (IP)," J. Postel, RFC-791, September 1981.

[IP:2] "Internet Control Message Protocol (ICMP)," J. Postel, RFC-792, September 1981.

[IP:3] "Internet Standard Subnetting Procedure," J. Mogul and J. Postel, RFC-950, August 1985.

[IP:4] "Host Extensions for IP Multicasting," S. Deering, RFC-1112, August 1989.

[IP:5] "Military Standard Internet Protocol," MIL-STD-1777, Department of Defense, August 1983.

This specification, as amended by RFC-963, is intended to describe the Internet Protocol but has some serious omissions (e.g., the mandatory subnet extension [IP:3] and the optional multicasting extension [IP:4]). It is also out of date. If there is a conflict, RFC-791, RFC-792, and RFC-950 must be taken as authoritative, while the present document is authoritative over all.

[IP:6] "Some Problems with the Specification of the Military Standard Internet Protocol," D. Sidhu, RFC-963, November 1985.

[IP:7] "The TCP Maximum Segment Size and Related Topics," J. Postel, RFC-879, November 1983.

Discusses and clarifies the relationship between the TCP Maximum Segment Size option and the IP datagram size.

[IP:8] "Internet Protocol Security Options," B. Schofield, RFC-1108, October 1989.

[IP:9] "Fragmentation Considered Harmful," C. Kent and J. Mogul, ACM SIGCOMM-87, August 1987. Published as ACM Comp Comm Review, Vol. 17, no. 5.

This useful paper discusses the problems created by Internet fragmentation and presents alternative solutions.

[IP:10] "IP Datagram Reassembly Algorithms," D. Clark, RFC-815, July 1982.

This and the following paper should be read by every implementor.

[IP:11] "Fault Isolation and Recovery," D. Clark, RFC-816, July 1982.

SECONDARY IP REFERENCES:

[IP:12] "Broadcasting Internet Datagrams in the Presence of Subnets," J. Mogul, RFC-922, October 1984.

[IP:13] "Name, Addresses, Ports, and Routes," D. Clark, RFC-814, July 1982.

[IP:14] "Something a Host Could Do with Source Quench: The Source Quench Introduced Delay (SQUID)," W. Prue and J. Postel, RFC-1016, July 1987.

This RFC first described directed broadcast addresses. However, the bulk of the RFC is concerned with gateways, not hosts.

UDP REFERENCES:

[UDP:1] "User Datagram Protocol," J. Postel, RFC-768, August 1980.

TCP REFERENCES:

[TCP:1] "Transmission Control Protocol," J. Postel, RFC-793, September 1981.

[TCP:2] "Transmission Control Protocol," MIL-STD-1778, US Department of Defense, August 1984.

This specification as amended by RFC-964 is intended to describe the same protocol as RFC-793 [TCP:1]. If there is a conflict, RFC-793 takes precedence, and the present document is authoritative over both.

[TCP:3] "Some Problems with the Specification of the Military Standard Transmission Control Protocol," D. Sidhu and T. Blumer, RFC-964, November 1985.

[TCP:4] "The TCP Maximum Segment Size and Related Topics," J. Postel, RFC-879, November 1983.

[TCP:5] "Window and Acknowledgment Strategy in TCP," D. Clark, RFC-813, July 1982.

[TCP:6] "Round Trip Time Estimation," P. Karn & C. Partridge, ACM SIGCOMM-87, August 1987.

[TCP:7] "Congestion Avoidance and Control," V. Jacobson, ACM SIGCOMM-88, August 1988.

SECONDARY TCP REFERENCES:

[TCP:8] "Modularity and Efficiency in Protocol Implementation," D. Clark, RFC-817, July 1982.

[TCP:9] "Congestion Control in IP/TCP," J. Nagle, RFC-896, January 1984.

[TCP:10] "Computing the Internet Checksum," R. Braden, D. Borman, and C. Partridge, RFC-1071, September 1988.

[TCP:11] "TCP Extensions for Long-Delay Paths," V. Jacobson & R. Braden, RFC-1072, October 1988.

Security Considerations

There are many security issues in the communication layers of host software, but a full discussion is beyond the scope of this RFC.

The Internet architecture generally provides little protection against spoofing of IP source addresses, so any security mechanism that is based upon verifying the IP source address of a datagram should be treated with suspicion. However, in restricted environments some source-address checking may be possible. For example, there might be a secure LAN whose gateway to the rest of the Internet discarded any incoming datagram with a source address that spoofed the LAN address. In this case, a host on the LAN could use the source address to test for local vs. remote source. This problem is complicated by source routing, and some have suggested that source-routed datagram forwarding by hosts (see Section 3.3.5) should be outlawed for security reasons.

Security-related issues are mentioned in sections concerning the IP Security option (Section 3.2.1.8), the ICMP Parameter Problem message (Section 3.2.2.5), IP options in UDP datagrams (Section 4.1.3.2), and reserved TCP ports (Section 4.2.2.1).

Author's Address

Robert Braden
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

Phone: (213) 822 1511

EMail: Braden@ISI.EDU